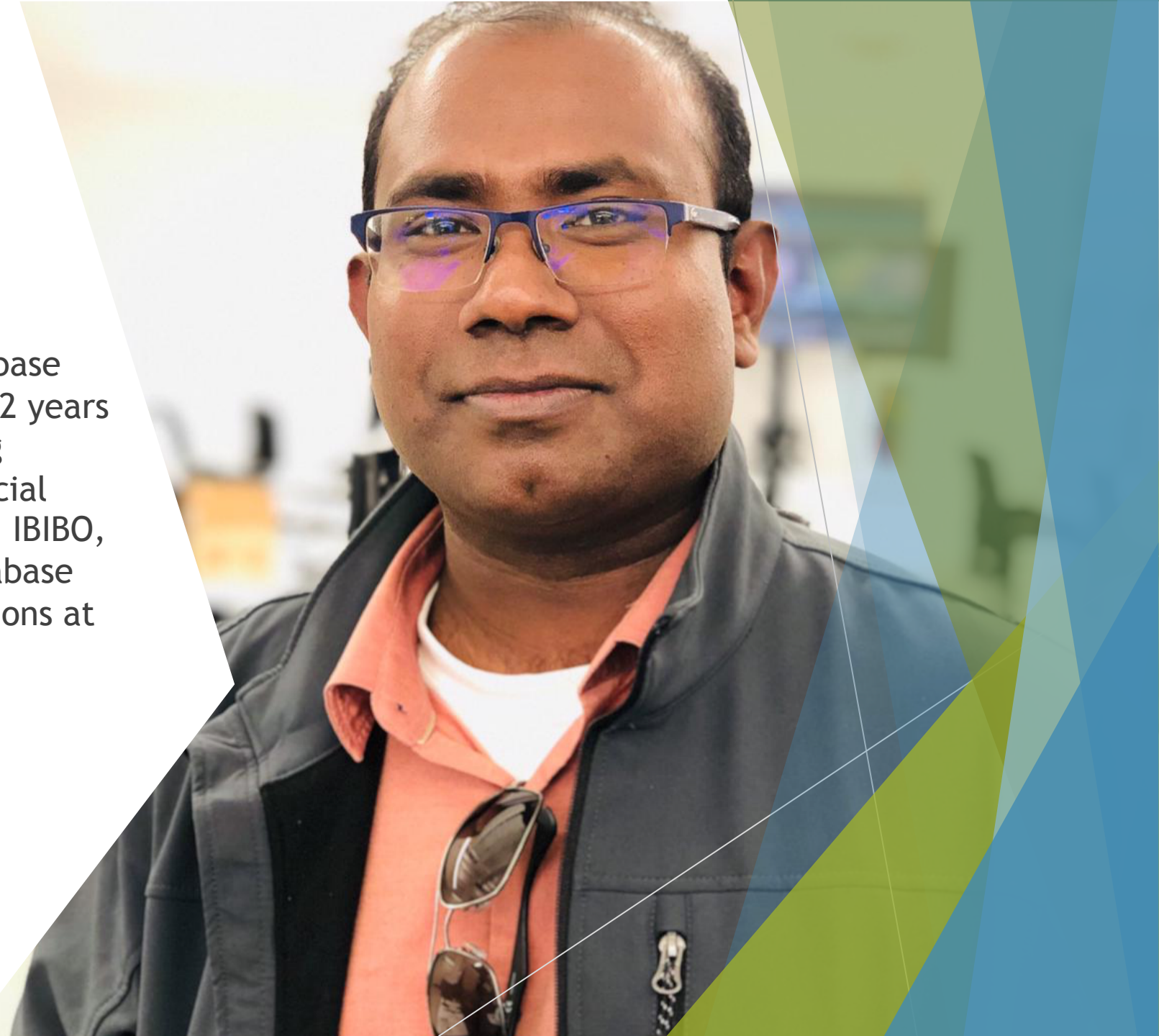
Gtid Concepts , Implementation and Challenges

Who am I ?

- Santhinesh Kumar Nagendran
- Currently working as Senior Database Administrator @ Tesla Inc. Over 12 years Industry experience in supporting environments like healthcare, social networking applications like AOL, IBIBO, Sify etc. I primarily focus on Database High availability and DB automations at large scale.




Concepts
Implementation
Challenges in Implementations and Operations

Agenda

Concepts

GTID Concepts

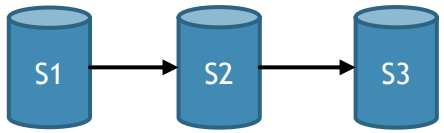
- A global transaction identifier (GTID) is a unique identifier created for each transaction committed on the server. This identifier is unique not only to the server on which it originated but is unique across all servers in a given replication cluster.
- A GTID is represented as a pair of coordinates, separated by a colon character (:), as shown here:
- GTID = *Server_uuid* :*transaction_id*

Principles of GTID

- auto-skip functionality in GTID ensures that transactions are not applied twice.
- A transaction which has started with a gtid on a server but did not commit or rollback, any attempt to start a concurrent transaction with the same GTID will block

Drawbacks of Binlog Positioning

- Getting Master position will be a complicated procedure for replication topology as below
- For an example as below, S denotes Server1/2/3



- ✓ *Master position for S2 will be different than the master position for S3*
- ✓ *The backup with binlog position of S1 cannot be easily used to build S3 as slave of S2. However, we can still achieve that using below procedure*
  - ✓ *We can still do this by building S3 slave of S1*
  - ✓ *Do an exercise to make S2/S3 to stop at same position*
  - ✓ *Get master status from S2*
  - ✓ *Run change master in S3 with the coordinates collected from S2 in above step*

- So the bottom-line is setting up a simple slave setup becomes a complex thing with binlog positioning if you don't have the backup from right source

Benefits of Auto Positioning

- Change master with master_auto_position=1 can be run with any host as master in the cluster and replication will just work fine
- Don’t have to deal with the complexity involved for finding the correct position from the binlog to form the change master statement
- Backup from one source can be used to setup replication from any host in the cluster
- Master/Slave Failover becomes easy just with one command
- Life improvisation for DBAs ☺

gtid_executed

- Primary Node – Gtids of transactions committed gets updated in gtid_executed
- Secondary/Slave nodes – Stores the latest transaction slave have applied
- From 5.7 and later gtid_executed gets stored in a system table mysql.gtid_executed

gtid_purged

- Primary Node – Gtids of the transactions that mysql has committed and corresponding binary log has been purged

What sets GTID_PURGED system variable globally ??

- When binary logs has been purged manually or as a regular event due to expire_logs_days value
- GTID_PURGED can be set explicitly by the below statement . Only prerequisite is the server should be in read-only mode to not accept any writes when we do reset master.
  - RESET MASTER;
  - SET @@GLOBAL.gtid_purged='gtid-set';

Every binlog starts with the previous GTIDs as shown in the example below.

```
# at 123
#181231 22:56:32 server id 103348190  end_log_pos 194 CRC32 0x7ba6b873  Previous-GTIDs
# 595b698b-c03c-11e8-9664-005056935aad:1-10764627
```

gtid_purged Cont..

```
mysql> show binary logs
    -> ;
+------------------+------------+
| Log_name         | File_size  |
+------------------+------------+
| mysql-bin.000007 | 1073742167 |
| mysql-bin.000008 |  841382931 |
+------------------+------------+
2 rows in set (0.00 sec)

mysql> show global variables like '%gtid_purged%'\G
*************************** 1. row ***************************
Variable_name: gtid_purged
        Value: 595b698b-c03c-11e8-9664-005056935aad:1-8522921
1 row in set (0.00 sec)

mysql> purge binary logs to 'mysql-bin.000008';
Query OK, 0 rows affected (0.42 sec)

mysql> show global variables like '%gtid_purged%'\G
*************************** 1. row ***************************
Variable_name: gtid_purged
        Value: 595b698b-c03c-11e8-9664-005056935aad:1-10764627
1 row in set (0.00 sec)
```

Implementation

GTID Implementation

- 5.6
- 5.7
- 8.0

Common parameters

gtid_mode	-	Will generate GTIDs for all client transactions
enforce_gtid_consistency	-	Allows only transactionally safe transactions

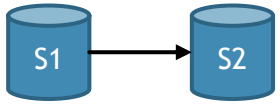
Transactionally un-safe statements

Before enabling GTID make sure application is not having transactionally unsafe statements as below. Otherwise this will cause unnecessary outages.

1. CREATE TABLE ... SELECT statements
2. CREATE TEMPORARY TABLE statements inside transactions
3. Transactions or statements that update both transactional and nontransactional tables.

Implementation With Downtime

Let's take an environment in which [master – Slave] are already in synchronize so the gtid enabling process will be as below



- Set read-only ON on both the servers
- Shutdown MySQL
- Restart both servers with below parameters in the config file along with disable statements not compatible for GTID-based replication.

```
[mysqld]
skip-slave-start
read_only                       = ON
gtid_mode                       = ON
enforce-gtid-consistency        = ON
log-slave-updates               = 1
binlog_format                   = row
log-bin                         =/mysql/logs/mysql-bin
```

- Instruct the slave to use the master as the replication data source and to use auto-positioning.
  - Change master to master_auto_position=1;
- Take a new xtrabackup or mysqldump or mysqlbackup as the backups taken until this migration are unusable hereafter
- Start the slave, then disable read-only mode again on primary/master servers, so that it can accept updates.

GTID Modes

Gtid modes	Functionality
OFF	- Master server will not generate any GTIDs for any client connections - If the Server is a replication slave it will only accept transactions without GTID
OFF_PERMISSIVE	- Servers will not generate any GTIDs for any client connections - However, If the Server is a replication slave it will accept transactions with and without GTID
ON_PERMISSIVE	- Master server will generate GTIDs for any new client connections - However, If the Server is a replication slave it will only accept transactions without GTID
ON	- Master server will generate GTIDs for all client connections - If the Server is a replication slave it will only accept transactions with GTID

Online Implementation

Let's continue with the same environment with gtid_mode as OFF across the whole cluster.

1. Validate if all the servers are in non-gtid enabled state.
2. Keeping server to not generate GTID for new transactions but it enables servers to accepts transactions with and without GTID
3. Configure servers to generate GTID for new transactions and retain the forgiving behavior of accepting transactions with and without GTID
4. Last stage is to enforce servers to generate GTID for new transactions and also accept transactions only with GTID

Online Implementation Cont..

Exact commands to enable GTID transactions:

Prerequisite

- SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = WARN;
  - If your application queries are suitable for gtid then you shouldn't be seeing any errors in the log

GTID Enabling sequence

1. SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = ON;
2. SET @@GLOBAL.GTID_MODE = OFF_PERMISSIVE;
3. SET @@GLOBAL.GTID_MODE = ON_PERMISSIVE;
4. SHOW STATUS LIKE 'ONGOING_ANONYMOUS_TRANSACTION_COUNT';
   - make sure all the transactions got replicated till step 5
5. SET @@GLOBAL.GTID_MODE = ON;

Challenges

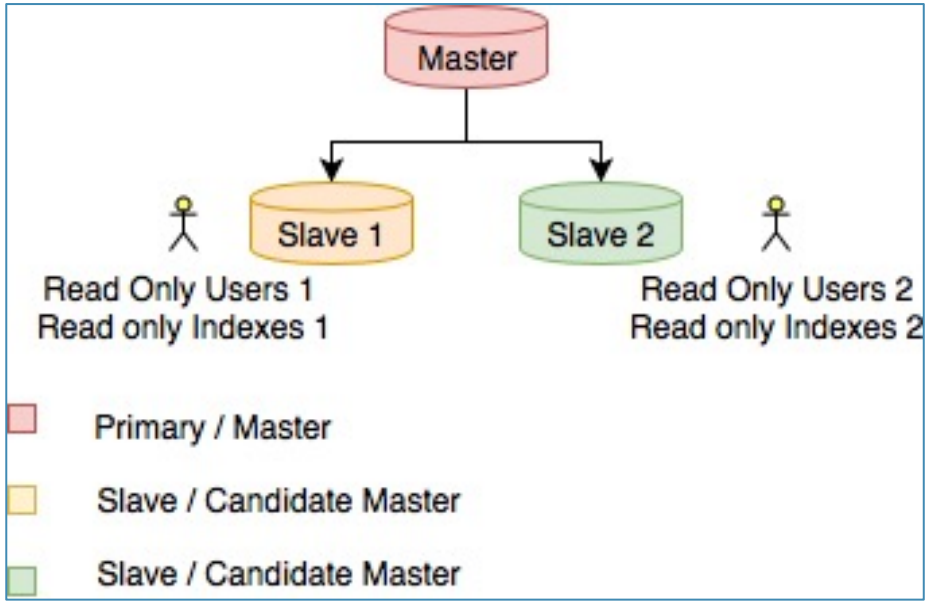
Replication Issues [Very Common]

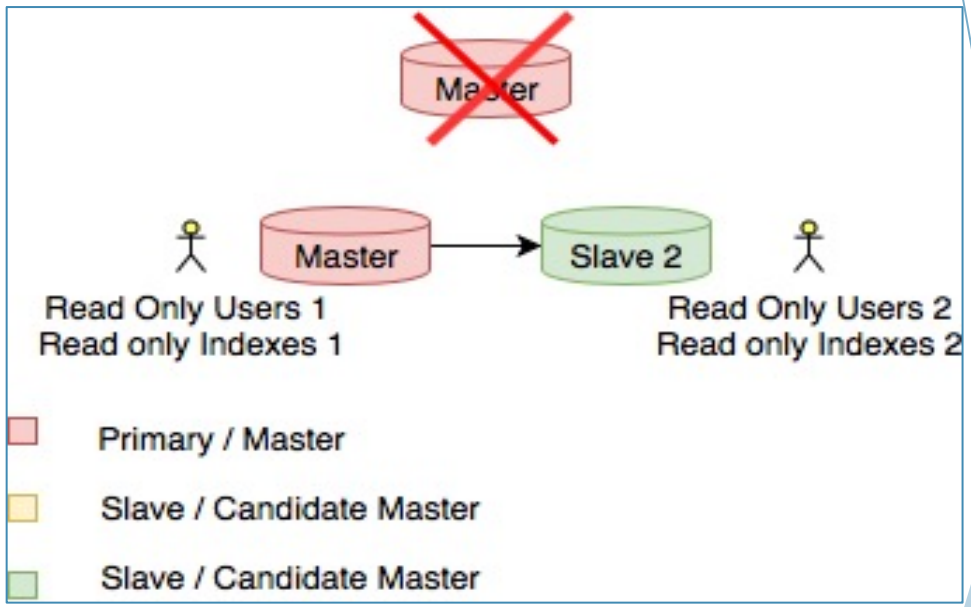
Last_IO_Errno: 1236

Last_IO_Error: Got fatal error 1236 from master when reading data from binary log: 'The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION = 1, but the master has purged binary logs containing GTIDs that the slave requires.'

- Master purges the binlog ***[accidentally or planned]*** before slaves caught up
- Event of failover to a slave and slave might had some local transactions which other slaves in the cluster might have not been aware of. Example :
  - Specific Indexes on slaves to help reads
  - Specific users on slaves alone for reporting and read-only purposes
  - Specific reporting or analytical data on slave

3 Node Replication Setup





Inconsistency Scenarios

- Slave 1 / Master will be unaware of users/properties created @ slave 2
- Slave 2 / Master will be unaware of users/properties created @ slave 1
- These users might have got created over the period of time on the slaves for which the binlogs also might have already got purged.
- So when failover happens, we would promote one of the slaves which is most up to date with the master
- So with the above conditions when we try to auto-position slave 2 to replicate from slave 1 where slave 2 is completely unware of transactions happened on slave 1 and slave 1 also purged the binlogs associated with such transactions we would encounter the error I mentioned above

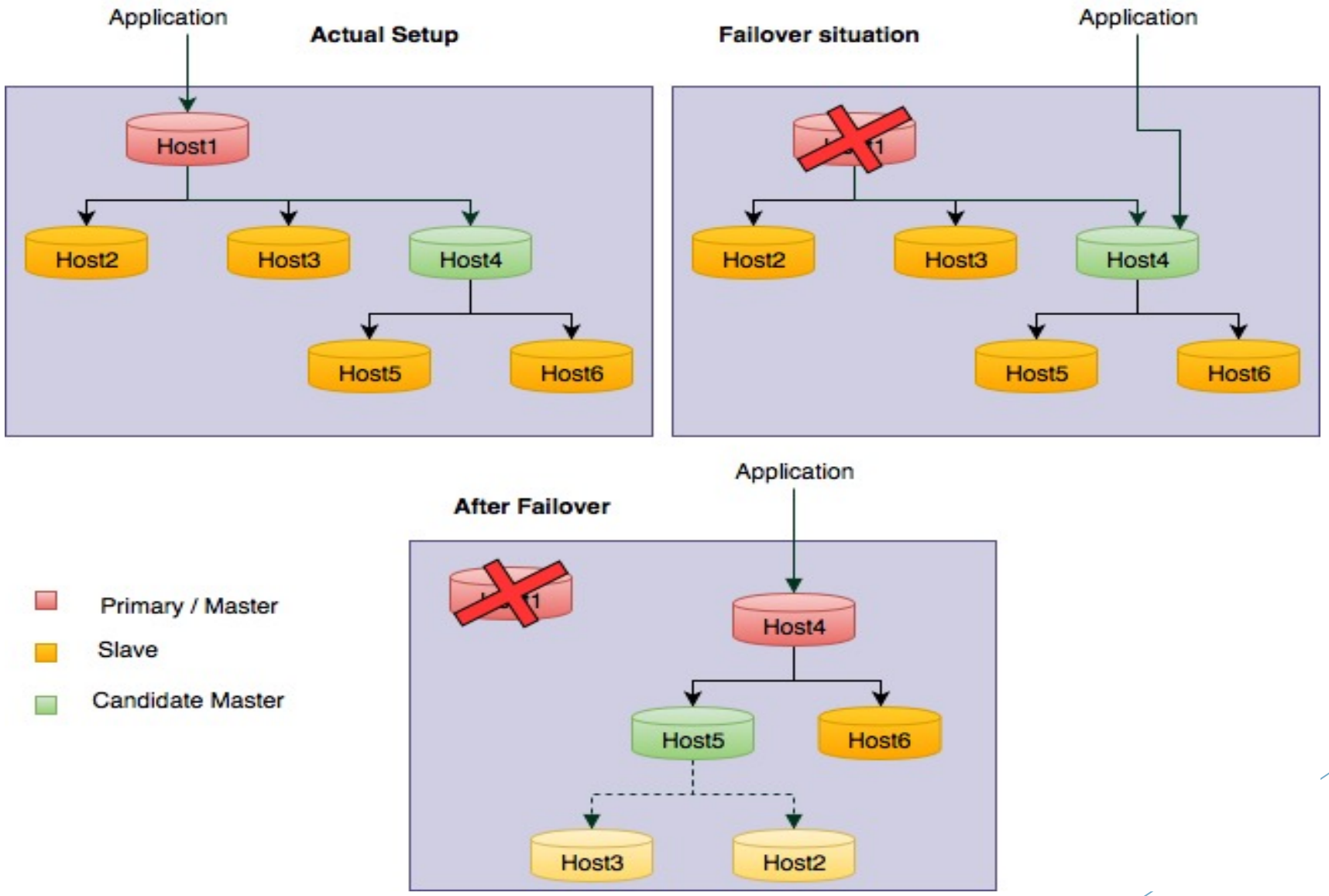
Challenges

Here are few ways to fix the inconsistency discussed earlier

- Gtid_purged cannot be set without RESET MASTER until 8.0
- Any Unplanned or accidental RESET MASTER on a critical master/slave will break the whole cluster
- Set GTID purged on the slave [using there most recent gtid_executed by itself] + [GTIDs which master has purged]
- Find the missing GTIDs on the slave that is broken
- Skip the GTIDs that slave is looking for which master has already purged.

NOTE: Beware sql_slave_skip_counter will not work with GTID based replication

Use Case


Application
Actual Setup
Host1
Host2
Host3
Host4
Host5
Host6
Failover situation
Application
Host2
Host3
Host4
Host5
Host6
Application
After Failover
Host4
Host5
Host6
Host3
Host2
Primary / Master
Slave
Candidate Master

Pointing Host2/Host3 to new Master Host4

Host4 – Master

GTID_PURGED :

0923xxxx-40c2-11e7-915d-0050569ade41:1-173383470,
1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,
3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,
3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,
491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,
8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,
67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-6426609,
e187xxxx-65f5-11e8-bc96-1866daa373fc:1-19,
3703xxxx-7816-11e8-8db9-000af7ba6042:1-8

Server - Host2
Slave Master - Host1

GTID_EXECUTED :

0923xxxx-40c2-11e7-915d-0050569ade41:1-173383470,
1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,
3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,
3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,
491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,
8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,
67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-16869434,
e187xxxx-65f5-11e8-bc96-1866daa373fc:1-27
b542xxxx-6079-11e8-9e44-1866daa37e46:1-8142,

Pointing Replication to new Master

Host4 – Master

GTID_PURGED :

~~0923xxxx-40c2-11e7-915d-0050569ade41:1-173383470,~~
~~1a02fxxxx-000a-11e8-a2c6-0050569a481a:1-39,~~
~~3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,~~
~~3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,~~
~~491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,~~
~~8caefxxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,~~
67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-6426609,
e187xxxx-65f5-11e8-bc96-1866daa373fc:1-19,
3703xxxx-7816-11e8-8db9-000af7ba6042:1-8

Server - Host2
Slave Master - Host1

GTID_EXECUTED :

~~0923xxxx-40c2-11e7-915d-0050569ade41:1-173383470,~~
~~1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,~~
~~3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,~~
~~3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,~~
~~491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,~~
~~8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,~~
67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-16869434,
e187xxxx-65f5-11e8-bc96-1866daa373fc:1-27
b542xxxx-6079-11e8-9e44-1866daa37e46:1-8142,

Current master - Host1
New Master - Host4

set global gtid_purged="0923xxxx-40c2-11e7-915d-0050569ade41:1-173383470,
1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,
3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,
3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,
491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,
8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,
67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-16869434,
e187xxxx-65f5-11e8-bc96-1866daa373fc:1-27,
3703xxxx-7816-11e8-8db9-000af7ba6042:1-8";

Pointing Replication to new Master Cont.

Run this in Host2

- ✓ stop slave;
- ✓ reset master;
- ✓ set global gtid_purged="

0923xxxx-40c2-11e7-915d-0050569ade41:1-173383470,
1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,
3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,
3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,
491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,
8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,
67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-16869434,
e187xxxx-65f5-11e8-bc96-1866daa373fc:1-27,
3703xxxx-7816-11e8-8db9-000af7ba6042:1-8";

- ✓ change master to master_host='host4',master_port=3306,master_auto_position=1;
- ✓ start slave;

GTID_SUBTRACT

```
mysql> select gtid_subtract('092311dc-40c2-11e7-915d-
0050569ade41:1-173383470,
    '> 1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,
    '> 3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,
    '> 3703xxxx-7816-11e8-8db9-000af7ba6042:1-8,
    '> 3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,
    '> 491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,
    '> 67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-6426609,
    '> 8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,
    '> e187xxxx-65f5-11e8-bc96-1866daa373fc:1-19',
    -> '092cxxxx-40c2-11e7-915d-0050569ade41:1-173383470,
    '> 1a02xxxx-000a-11e8-a2c6-0050569a481a:1-39,
    '> 3079xxxx-892a-11e7-bef9-0050569a2e78:1-17,
    '> 3f90xxxx-fd7f-11e7-b787-0050569a4168:1-201198177,
    '> 491axxxx-136e-11e8-b54d-000af7ba704a:1-1351372450,
    '> 67a8xxxx-5a5d-11e8-b3f2-000af7ba4704:1-16869434,
    '> 8caexxxx-2f25-11e8-a6fd-1866daa3d37d:1-936,
    '> b542xxxx-6079-11e8-9e44-1866daa37e46:1-8142,
    '> e187xxxx-65f5-11e8-bc96-1866daa373fc:1-27') as
missing_gtid_set;
+-----------------------------------------+
| missing_gtid_set                        |
+-----------------------------------------+
| 3703xxxx-7816-11e8-8db9-000af7ba6042:1-8 |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Empty Transactions

Generate Empty transactions for the missing GTIDs in slave there is a script shared by percona team on the below reference

```
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:1';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:2';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:3';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:4';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:5';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:6';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:7';
BEGIN;COMMIT;
SET GTID_NEXT='3703xxxx-7816-11e8-8db9-000af7ba6042:8';
BEGIN;COMMIT;
```

Reference : https://www.percona.com/blog/2018/07/02/fixing-er_master_has_purged_required_gtids-when-pointing-a-slave-to-a-different-master/

mysqlslavetrx

Injecting Single GTIDs set

```
mysqlslavetrx --gtid-set=3703xxxx-7816-11e8-8db9-000af7ba6042:1-8 --verbose --
slaves=root:password@localhost:3306

WARNING: Using a password on the command line interface can be insecure.
#
# GTID set to be skipped for each server:
# - localhost@3306: 46fdxxxx-5852-11e6-92c9-0800274fb806:1-8
#
# Injecting empty transactions for 'localhost:3306'...
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:1
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:2
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:3
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:4
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:5
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:6
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:7
# - 3703xxxx-7816-11e8-8db9-000af7ba6042:8
#
#...done.
#
```

Injecting Multiple empty GTIDs sets

```
mysqlslavetrx --gtid-set=3703xxxx-7816-11e8-8db9-000af7ba6042:9-12,8caexxxx-2f25-
11e8-a6fd-1866daa3d37d:936-945 --verbose --slaves=root:password@localhost:3306
```

Is RESET MASTER a Scary Option ? Then what else ?

Best way to deal with GTID is by NOT GETTING PANIC at any point of time

1) Collect the show slave status
2) Collect global variables value for gtid_purged and gtid_executed.
3) Gtid_executed has the last position the slave was able to apply correctly
4) Gtid_purged has the [starting position -1]
5) All we need is higher gtid_executed values than gtid_purged
6) **gtid_subtract (master_gtid_purged - slave_gtid_executed)** will give the gtids missing in slave
7) Generate empty transactions for the gtids missing obtained using gtid_subtract using mysqlslavetrx as showed before or use a shell script to generate empty transactions
8) Once empty transactions are loaded start replication. It should fix the issue
9) Don’t do RESET MASTER if you don’t know what else to do instead Call OUT for help !!!!
10) Use Pt-table-checksum to identify inconsistency between master/Slave.

Conclusion

- GTID is one of The best features in MySQL.
- With the enhancements in MySQL 8.0 which addressed most of the limitations around setting up gtid_purged at runtime makes its even more powerful to be implemented
- Enabling and stabilizing the environment for the first time might be a little bit of task but in long run it would eventually save lots of operation time and efforts

Thanks for listening!
Any questions?