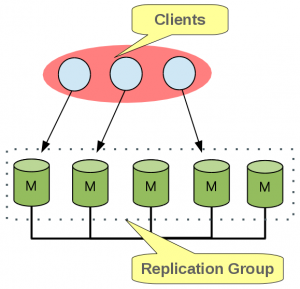
MySQL Group Replication – Transaction life cycle explained

The multi master plugin for MySQL is here. MySQL Group Replication provides virtually synchronous updates on any member in a group of MySQL servers, with conflict handling and automatic group membership management and failure detection.

For a better understanding on how things work, we go under the hood in this post and will analyze the transaction life cycle on multi master and which components does it interact with. But before that we need to understand first what a group is.

### **Group Communication Toolkit**

The multi master plugin is powered by a group communication toolkit. This is what decides which servers belong to the group, performs failure detection and orders server messages. Being the ordered messaging the magic thing that allows the data to be consistent across all members. You can check the details of the group communication toolkit at [Group communication behind the scenes](http://mysqlhighavailability.com/group-communication-behind-the-scenes/) post.

[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/1.png)

These strong properties, together with the same initial state of each server that joins the group, allowed us to implement a database state machine replication (DBSM)[1].

On the context of group communication system other of the base concepts is the view. When a server joins (joiner) the group, a new view is installed, being this view a logical marker that defines which servers belong to the group. A view is also installed when a server leaves the group (either voluntarily or involuntarily). This is provided by a built-in  
dynamic group membership service.

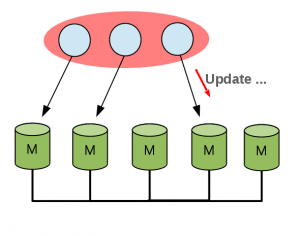
Before start applying transactions, joiners will request from the group its current state, in order to catch up missing transactions, thus synchronizing to an up-to-date replica. This is called Distributed Recovery. You can check more details at [Distributed Recovery behind the scenes](http://mysqlhighavailability.com/distributed-recovery-behind-the-scenes/) post.

### **Transaction life cycle**

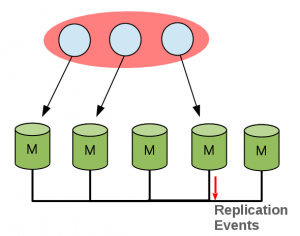
Having the group set up and the same initial state on all group members, now we need that all group members agree on transactions outcome, that is, if any transaction is committed on any member it must also be committed on all the others non-faulty group members.  
This strong requirement is only needed for write operations, read-only transactions are executed locally only.

Group communication toolkit provides us a total order broadcast primitive, that is, all messages are delivered everywhere on the same order even in the presence of failures. This means that we have a global order of messages exchanged on the group, combined together with deterministic transaction outcome (commit or rollback) decisions can ensure that all servers eventually will reach the same state.

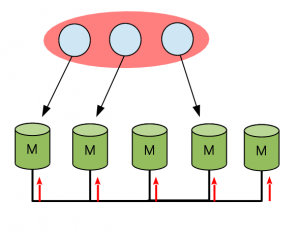
So, whenever a client executes a write transaction, it is executed optimistically on the local server right until before being actually committed.

[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/2.png)

At this stage the transaction is broadcast to the group so that its outcome is decided.

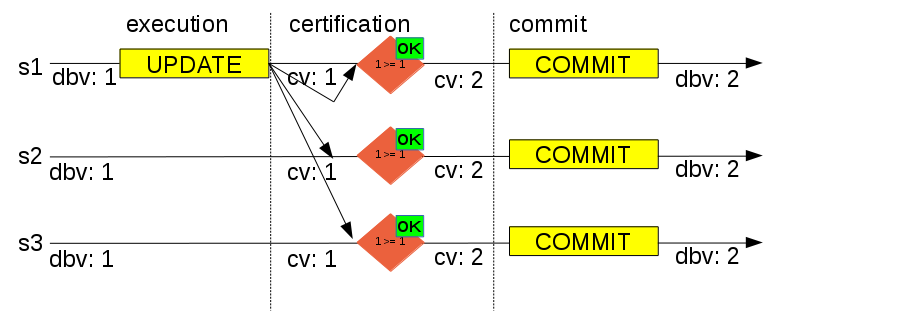
[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/3.png)

This message is ordered by group communication toolkit and delivered to all non-faulty group members in same order.

[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/4.png)  
Now we need to decide locally if this particular transaction should be committed or rollback. Every server keeps for each row update an associated version. This will allow us to know that the version of the row that serves as base for this update is an older version and so these where executed in parallel in different members. This information about the version at which a given transaction was executed, is included in transaction write set.

### **Certification**

This decision is taken by the certification module. Let’s see an example of it.

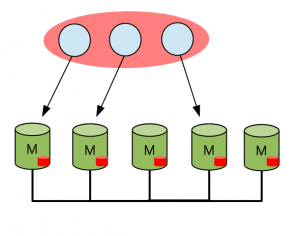
[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/5.png)

On the above diagram we have a group composed by three servers, on which a client will execute a write transaction on server 1 (s1). The transaction executes until the before commit stage and then it broadcasts write set and data to the group.

The transaction write set is composed by the primary keys of each updated table row and the database version at which transaction was executed. Database version is provided by GTID\_EXECUTED, more precisely its last number without gaps from group UUID.  
Example:  GTID\_EXECUTED: UUID:1-10, UUID:12, version is 10.  
Database version is implicitly increased with transactions commit.

After delivery, the transaction is certified, that is, each server will compare locally the versions included on the write set with the local versions. If the version included on write set is smaller than the one on certification database then transaction rollbacks. No version at certification module for a particular row means that there was not an update for it yet and that this particular row won’t cause any transaction rollback.

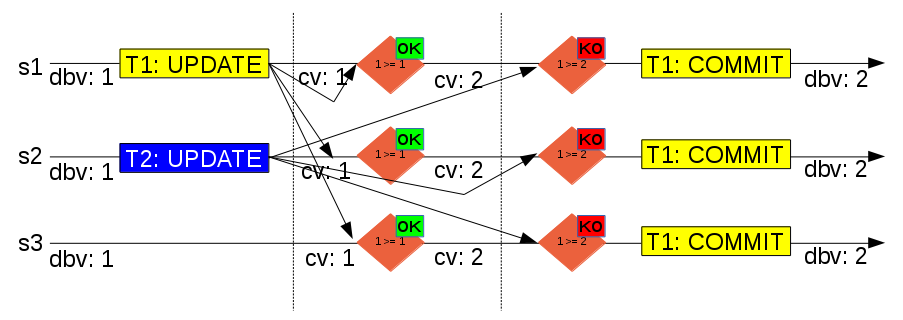
Database version during execution was 1 (dbv: 1) and current version at certification module is also 1 (cv: 1), which means that this transaction does not conflict with any other ongoing transaction. So, transaction will be allowed to commit, and row version on  
certification module updated to 2 (cv: 2). On the local server (s1) this means proceed to commit and return success to client. On the remote servers (s2 and s3) the transaction will be queued to be applied by applier module.  
Applier module is the responsible to apply positively certified transactions that arrived from the group, like server 2 and 3 on the above figure.

[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/6.png)

Transaction identifier (GTID) is controlled by the certification module, in order to a given transaction has the same identifier on all group members.  
All members of the group share the same UUID, so group behaves like a single server where all transactions have the same source UUID, which is the group name.  
More details at [Getting started with MySQL Group Replication](http://mysqlhighavailability.com/getting-started-with-mysql-group-replication/) post.

To achieve this, the certification module state is also transferred during distributed recovery when a server joins the group, again fulfilling the DBSM requirement that all servers on group must have the same initial state.

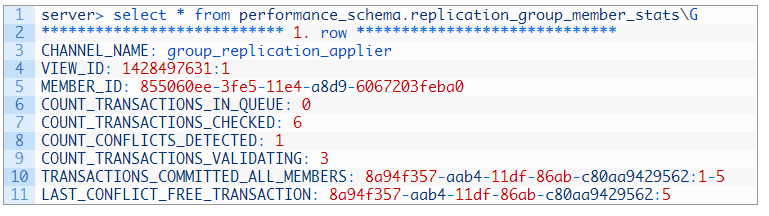
Let’s see an example on which we have conflicts between transactions.

[](http://mysqlhighavailability.com/wp-content/uploads/2014/09/7.png)

On the above diagram we have again a group with three servers, on which both server 1 (s1) and server 2 (s2) execute a transaction (T1 and T2) that updates the **same row** concurrently. Both transactions are executed locally until before commit stage on database version 1 (dbv: 1), and then the write set and write data are broadcast to the group.  
Group communication toolkit, on this example, ordered T1 before T2, so since certification module version for the update row is 1 (cv: 1), transaction will be positively certified, certification module row version will be updated to 2 (cv: 2), and the transaction will continue to commit.  
Then it is certification turn for T2, database version is again 1 (dbv: 1) but certification module row version is 2. This means that the changed row by this transaction was already updated by a transaction that happened after execution on server 2 and before certification (on all servers), so T2 must be negatively certified. Server 2 will roll back the transaction and return an error to the client, remote servers will discard the transaction.

One thing that you may be thinking is, will not certification module data increase forever? Without more implementation details, yes! But we have that covered, periodically all group members exchange their GTID\_EXECUTED to compute the intersection of the transactions that all servers already committed – the Stable Set. Any group member cannot update rows on version lower than that executed set. Then a garbage collector procedure is executed to remove all row versions that belong to the stable set.

Certification stats are available at performance\_schema replication\_group\_member\_stats table.



You can check more details about stats and monitoring at [Group Replication Monitoring](http://mysqlhighavailability.com/mysql-group-replication-testing/) post.

### **Conclusion**

On his first steps, MySQL Group Replication is still in development. On this blog post we explain the transaction life cycle on this new replication technology. Feel free to try it and get back at us so we can make it even better for the community!

<https://mysqlhighavailability.com/mysql-group-replication-transaction-life-cycle-explained/>

=====================================================================================

