**The magic of Galera replication**

The short of it is that Galera replication is not a doing a full 2-phase commit to replicate (where all nodes commit synchronously with the client), but it is instead something Codership calls “virtually synchronous” replication.  This could either be clever marketing or clever engineering, at least at face value.   However, I believe it really is clever engineering and that it is probably the best compromise for performance and data protection out there.

There’s likely a lot more depth we could cover in this definition, but fundamentally “virtually synchronous replication” means:

* Writesets (or “transactions”) are replicated to all available nodes in the cluster on commit (and enqueued on each).
* **EDIT:**Writesets are then “certified” on every node (in order).  This certification should be deterministic on every node, so every node either accepts or rejects the writeset.  There is no way for a node to tell the rest of the cluster a writeset didn’t pass certification (or this would be a form of two-phase commit), so the only way nodes might get different certification results is if there is a Galera bug.
* Enqueued writesets are applied on those nodes independently and asynchronously from the original commit on the source node.  And:
* At this point the transaction can and should be considered permanent in the cluster.  But how can that be true if they are not applied?  Because:
  + Galera can do conflict detection between different writesets, so enqueued (but not yet committed) writesets are protected from local conflicting commits until our replicated writeset is committed. AND:
  + When the writeset is actually applied on a given node, any locking conflicts it detects with open (not-yet-committed) transactions on that node cause that open transaction to get rolled back.
  + Writesets being applied by replication threads **always** win.

So why is this “virtually synchronous”?  Because simply getting our writesets to every node on commit means that they are guaranteed to apply — therefore we don’t have to force all nodes to commit simultaneously to guarantee cluster consistency as you would in a two-phase commit architecture.

**Seeing when replication conflicts happen**

This brings me to my topic for today, the mysterious SHOW GLOBAL STATUS variables called:

* wsrep\_local\_cert\_failures
* wsrep\_local\_bf\_aborts

I found that understanding these helped me understand Galera replication better.  If you are experiencing the “unexpected deadlocks” problem, then you are likely seeing one or both of these counters increase over time, but what do they mean?

Actually, they are two sides to the same coin (kind of).  Both apply to some local transaction getting aborted and rolled back, and the difference comes down to when and how that transaction conflict was detected.  It turns out there are two possible ways:

wsrep\_local\_cert\_failures

The [Galera documentation](http://www.codership.com/wiki/doku.php?id=galera_status_0.8" \t "_blank) states that this is the:

|  |
| --- |
| Total number of local transactions that failed certification test. |

What is a local certification test?  It’s quite simple:  On COMMIT, galera takes the writeset for this transaction and does conflict detection against all pending writesets in the local queue on this node.  If there is a conflict, the deadlock on COMMIT error happens (which shouldn’t happen in normal Innodb), the transaction is rolled back, and this counter is incremented.

To put it another way, some other conflicting write from some other node was committed before we could commit, and so we must abort.

This local certification failure is only triggered by a Galera writeset comparison operation comparing a given to-be-commited writeset to all other writesets enqueued locally on the local node.  The local transaction always loses.

**EDIT:** certification happens on every node.  A ‘local’ certification failure is only counted on the node that was the source of the transaction.

wsrep\_local\_bf\_aborts

Again, the [Galera documentation](http://www.codership.com/wiki/doku.php?id=galera_status_0.8" \t "_blank) states that this is the:

|  |
| --- |
| Total number of local transactions that were aborted by slave transactions while in execution. |

This kind of sounds like the same thing, but this is actually an abort from the opposite vector:  instead of a local transaction triggering the failure on commit, this is triggered by Galera replication threads applying replicated transactions.

To be clearer: a transaction was open on this node (not-yet-committed), and a conflicting writeset from some other node that was being applied caused a locking conflict.  Again, first committed (from some other node) wins, so our open transaction is again rolled back.  “bf” stands for brute-force:  any transaction can get aborted by galera any time it is necessary.

Note that this conflict happens only when the replicated writeset (from some other node) is being applied, not when it’s just sitting in the queue.  If our local transaction got to its COMMIT and this conflicting writeset was in the queue, then it should fail the local certification test instead.

A brute force abort is only triggered by a locking conflict between a writeset being applied by a slave thread and an open transaction on the node, not by a Galera writeset comparison as in the local certification failure.

**Testing it all out**

So this is the part of the post where I wanted to show that these counters were being incremented using an example from [my last post](https://www.percona.com/blog/2012/08/17/percona-xtradb-cluster-multi-node-writing-and-unexpected-deadlocks/).  [Those examples](https://www.percona.com/blog/2012/08/17/percona-xtradb-cluster-multi-node-writing-and-unexpected-deadlocks/) should trigger brute force aborts, but they didn’t seem to increment either of these counters on any of my testing nodes.   Codership agrees this seems like a bug and is investigating.  I’ll update this post if and when an actual bug is opened, but I have seen these counters being incremented in the wild, so any bug is likely some edge case.

By the way, I can’t think of how to reliably produce local certification errors without just a lot of fast modifications to a single row, because those depend on the replication queue being non-empty and I don’t know any way to pause the Galera queue for a controlled experiment.

<https://www.percona.com/blog/2012/11/20/understanding-multi-node-writing-conflict-metrics-in-percona-xtradb-cluster-and-galera/>