JEPSEN

MariaDB Galera Cluster 12.1.2

Kyle Kingsbury
2026-03-16

MariaDB Galera Cluster is a replicated version of MariaDB, a popular SQL database. While MariaDB claims Galera ensures "no lost transactions", it loses transactions in at least two scenarios. First, under the recommended configuration settings it does not flush data to disk before acknowledgement; committed transactions can be lost when nodes crash in quick succession. Second, it occasionally loses committed transactions with process crashes and network partitions. Even without faults, MariaDB Galera Cluster allows P4 (Lost Update), and therefore fails to satisfy its claimed isolation level "between Serializable and Repeatable Read". It also regularly exhibits Stale Read. This research was performed independently by Jepsen, without compensation, and conducted in accordance with the Jepsen ethics policy.

1 Background

MariaDB is a popular open source SQL database which began as a fork of MySQL. MariaDB Galera Cluster is an active-active replication system for MariaDB which allows reads and writes on every node. In 2015 this author analyzed MariaDB with Galera Cluster and found that although Galera claimed to offer Snapshot Isolation, Codership Oy intentionally designed their system without a key Snapshot Isolation property, called *first-committer-wins*. This allowed MariaDB with Galera Cluster to lose or create money out of thin air in a simulated bank account transfer workload. In 2025 MariaDB acquired Codership Oy, bringing Galera Cluster under the MariaDB umbrella.

Galera Cluster is based on virtual synchrony group communication framework called `gcomm`. Transactions are initially executed optimistically on any node. When a transaction commits it is synchronously replicated to other nodes, which certify the transaction based on the primary keys it wrote. Conflicts with other transactions are identified based on a sequence number, or seqno.

The MariaDB Galera Replication Guide says that Galera uses unanimous replication:

> Unlike traditional asynchronous or semi-synchronous replication, Galera ensures that transactions are committed on all nodes (or fail on all) before the client receives a success confirmation.

This is obviously wrong. If Galera actually required transactions to commit on all nodes, it would not tolerate a single node failure. MariaDB's documentation often repeats this claim, saying "a transaction is not truly considered committed until it has passed certification on all nodes", or "when a transaction `COMMIT`s, all nodes in the cluster have the same value", or "only after Node A gets an 'OK' from all other nodes does it tell the client, 'Your transaction is committed.'" In reality, Galera Cluster continues to operate when a minority of nodes has failed. This is consistent with MariaDB's claims about fault tolerance: if a quorum of nodes are online and connected, that component can make progress.[1]

Galera used to require a manual recovery procedure when quorum was lost: an operator would have to log in to every node, identify the node with the highest sequence number, and use it to bootstrap the cluster. However, newer versions of Galera can recover from failures automatically.

1.1 Safety

"Data is consistent across all nodes at all times," says the Galera Cluster Replication Guide, "preventing data loss upon node failures." Galera "essentially transforms a set of individual MariaDB servers into a robust, highly available, and consistent distributed database system."

This system should provide a real-time consistency model like Strong Snapshot Isolation. MariaDB's Galera Cluster Guide says that Galera Cluster's synchronous replication means that changes are "instantly replicated to all other nodes, ensuring no replica lag and no lost transactions." The "no lost transactions" claim is repeated in MariaDB's Galera Cluster README.

The Galera Cluster Usage Guide promises that "Standard SQL transactions (`START TRANSACTION`, `COMMIT`, `ROLLBACK`) work as expected." From this one might assume that MariaDB with Galera Cluster supports the same consistency models as a single MariaDB node. Is this true? It is surprisingly difficult to find out!

[1] Quorums are determined by tunable weights.

MariaDB's Galera documentation does include a section on known limitations. Some kinds of explicit locking are unsupported, and MariaDB must use the InnoDB storage engine. However, this list makes no mention of isolation levels or consistency anomalies. In fact, the sole reference to isolation levels Jepsen found in MariaDB's Galera documentation is buried in the Management section, under Installation and Deployment, on the Tips on Converting to Galera page, under the "Transaction size" heading.[2] It says:

> Galera's tx_isolation is between Serializable and Repeatable Read. tx_isolation variable is ignored.

Repeatable Read is a remarkably strong consistency model. In most formalisms it is equivalent to Serializability so long as objects are selected by primary key, rather than predicates. In MariaDB "Repeatable Read" used to allow non-repeatable reads but now prohibits them; per MDEV-35124, MariaDB "Repeatable Read" should actually provide Snapshot Isolation. We therefore expect MariaDB Galera Cluster to provide a consistency model weaker than Serializable, but stronger than Repeatable Read, Snapshot Isolation, or both.

2 Test Design

We adapted Jepsen's existing test suite for MySQL & MariaDB to set up three-node clusters of MariaDB with Galera Cluster, running on Debian Trixie. We used MariaDB's official Debian repositories to install MariaDB 12.1.2 through 12.2.2, and Galera 26.4.13 through 26.4.25. We used MariaDB's official Java client at version 3.5.6 to submit transactions to the cluster. While testing we introduced a variety of faults, including network partitions, process pauses, and process kills.

As in our previous MySQL analysis, our main workload used Elle's list-append checker for transactional isolation. In a nutshell, Elle infers Adya's write-write, write-read, and read-write dependencies between transactions, then looks for cycles in the resulting dependency graph, as well as a few other phenomena.

To infer these dependencies, our append workload performed randomly generated transactions over lists of integers, with each list identified by a unique primary key. Each micro-operation within a transaction could either read a list, or append a unique integer element to a list. As in previous work, we encoded these lists as a `text` column of comma-separated elements, and used SQL `concat` to append elements to a specific row. We split rows across multiple tables with a structure like:

```
create table "txn0" (
  id int not null primary key,
  val text
);
```

Since rows were only changed by appending a unique integer to its `val` column, any read of a row told Elle exactly which transactions wrote to it, and in which order. From this Elle inferred the version order for each row, which allowed inference of all three types of transaction data dependencies.[3] It also inferred session and real-time dependencies based on the concurrency structure of the recorded history. Elle then found strongly connected components in that graph, and searched for cycles with particular shapes to find counterexamples to a variety of consistency models. For example, a cycle involving only write-write and write-read edges would constitute G1c, a violation of Read Committed.

3 Results

3.1 Write Loss on Coordinated Process Crash (MDEV-38974)

When all nodes crashed at approximately the same time, MariaDB with Galera Cluster regularly lost committed transactions. For example, in this one-minute test run, the cluster lost nine values appended to three different rows. Reads of row 112 around the time of a process crash observed:

Time (s)	Elements
50.63	… 38, 45, 51, 53
50.64	… 38, 45, 51, 53, 56, 57, 58
50.64	… 38, 45, 51, 53, 56, 57
50.65	… 38, 45, 51, 53, 56, 57, 58
50.65	… 38, 45, 51, 53, 56, 57, 58
50.65	… 38, 45, 51, 53, 56, 57, 58, 71
50.65	… 38, 45, 51, 53, 56, 57, 58, 71
50.66	… 38, 45, 51, 53, 56, 57, 58, 71
65.73	… 38, 45, 51, 158, 159
65.73	… 38, 45, 51, 158, 159, 160

All of the transactions which wrote these values were acknowledged as successfully committed. However, when the cluster restarted the appends of 53, 56, 57, 58, and 71 to row 112 were lost, and new elements were appended in their place: 158, 159, 160, and so on. The lost elements never appeared in any later read.

This behavior seemed to be caused by setting `innodb_flush_log_at_trx_commit = 0`; setting it to `1` dramatically reduced the frequency of data loss. We initially chose `0` because MariaDB described it as "a safer, recommended option" in the documentation on Configuring MariaDB Galera Cluster:

[2] … in the bottom of a locked filing cabinet stuck in a disused lavatory with a sign on the door saying "Beware of the Leopard".

[3] Technically, we infer a mostly complete *prefix* of the version order, such that any "real" version order must be compatible with it. For more details, see the Elle paper.

`innodb_flush_log_at_trx_commit=0` — This is not usually recommended in the case of standard MariaDB. However, it is a safer, recommended option with Galera Cluster, since inconsistencies can always be fixed by recovering from another node.

This works when failures are uncoordinated, but coordinated failures do sometimes happen! Flooding, lightning, cooling, network bugs, and other failures can cause all nodes in a cluster to fail in rapid succession, and when this occurs, unsynced data can be lost. Jepsen reported this issue as MDEV-38974.

3.2 More Write Loss (MDEV-38976)

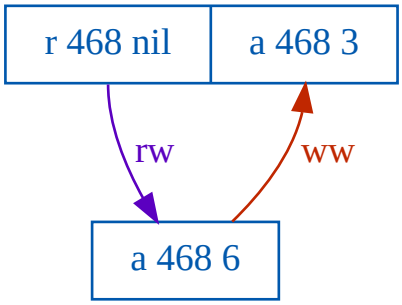
Setting `innodb_flush_log_at_trx_commit=1` significantly reduced data loss, but did not eliminate it. Infrequently, MariaDB Galera Cluster lost the effects of committed transactions when tests involved process crashes and network partitions. For example, at roughly 141 seconds into this test run, the cluster lost approximately nineteen seconds of writes across four separate objects: `0`, `285`, `410`, and `446`. Some, like key `0`, lost only a short postfix of elements. Key `410`, on the other hand, lost all twenty-five elements and began afresh:

Time (s)	Elements
141.36	17, 19, 26, …, 91, 92, 97
152.79	175
153.21	175, 176, 177, 179
154.46	175, 176, 177, 179, 180

Note that the transactions which wrote `17`, `19`, and so on were successfully committed; their effects definitely should not have been lost. This issue appeared only once every few hours of testing, and seems unlikely to affect production users. Nevertheless the loss of committed writes is concerning, and Jepsen reported this to MariaDB as MDEV-38976.

3.3 Lost Update (MDEV-38977)

Even when write loss did not occur, Galera Cluster allowed P4 (Lost Update) and other forms of G-single. These anomalies occurred even in healthy clusters, without faults. For example, consider this test run, which contained the following pair of transactions:



The top transaction read key `468` and found nothing, then appended `3` to it. The bottom transaction appended `6` to key `468`. However, later reads of key `468` all found values beginning with `[6, 3, ...]`, which implies that the bottom transaction (apparently) modified key 468 *between* the top transaction's read and write of it. This is a straightforward example of Lost Update, from Berenson et al.'s paper defining Snapshot Isolation:

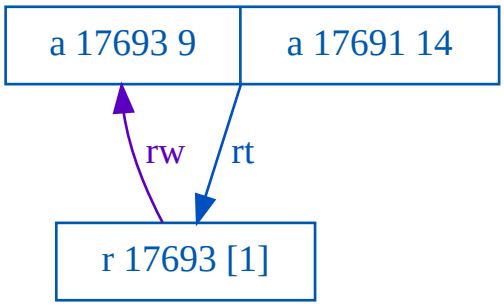
> **P4 (Lost Update)**: The lost update anomaly occurs when transaction T1 reads a data item and then T2 updates the data item (possibly based on a previous read), then T1 (based on its earlier read value) updates the data item and commits. In terms of histories, this is:
>
> **P4**: r1[x]…w2[x]…w1[x]…c1.
>
> The problem … is that even if T2 commits, T2's update will be lost.

P4 violates Snapshot Isolation. Since all operations here involved access by primary key, rather than predicates, this cycle is also G2-item: a violation of Repeatable Read.

We also observed more complex cycles involving multiple keys, or more than two transactions, all of which had a single read-write dependency edge. These cycles are examples of G-single, which is a more general violation of Snapshot Isolation. They similarly violate Repeatable Read. Jepsen reported this issue as MDEV-38977.

3.4 Stale Read (MDEV-38999)

Finally, under normal operation, MariaDB Galera Cluster occasionally allowed Stale Reads: a transaction could commit, be acknowledged as successfully completed to the client, then a second transaction could begin and fail to observe the first transaction's writes. For example, take this test run, which contained the following pair of transactions:



The top transaction appended 9 to key 17693, then committed and was acknowledged to the client. The bottom transaction began after that acknowledgement, hence the real-time (*rt*) dependency edge from top to bottom. However, the bottom transaction read key 17963, and failed to observe the top transaction's append of 9; hence the read-write (*rw*) dependency. This is a stale read, which is inconsistent with Galera Cluster's claims of instant, lag-free replication.

This behavior occurred every few minutes in our testing, even without fault injection. Jepsen reported this issue to MariaDB as MDEV-38999.

№	Summary	Event Required	Fixed in
MDEV-38974	Loss of committed writes	Coordinated process crashes	Unresolved
MDEV-38976	Loss of committed writes	Process crashes and network partitions	Unresolved
MDEV-38977	Lost Update	None	Unresolved
MDEV-38999	Stale Read	None	Unresolved

4 Discussion

MariaDB Galera Cluster claimed to offer an isolation level "between Serializable and Repeatable Read", and that transactions were "instantly replicated to all other nodes, ensuring no replica lag and no lost transactions". However, when configured with MariaDB's recommended settings, it lost committed transactions when multiple nodes failed in rapid succession. It also occasionally lost committed transactions under process crashes and network partitions. Even in healthy clusters, MariaDB Galera Cluster exhibited Lost Update and Stale Read; it provided neither Snapshot Isolation nor Repeatable Read, nor their stronger real-time variants. Indeed, the loss of committed transactions suggests MariaDB Galera Cluster was weaker than Read Uncommitted.

Users should set `innodb_flush_log_at_trx_commit=1` to reduce the probability of write loss on coordinated failure. MariaDB should revise their documentation to make it clear that changing this setting to `0` allows data loss in Galera Cluster.

Even with `innodb_flush_log_at_trx_commit=1`, users should expect MariaDB Galera Cluster to lose committed writes when node failures and network partitions occur. Thankfully, this behavior does not appear to be common. It also exhibits Stale Read, Lost Update, and other forms of G-single in healthy clusters, when no faults occur. Transactions may (apparently) modify data in the interval between a single transaction's reads and writes; read-modify-write patterns, like those used in many ORMs, are likely unsafe. Users should also assume that committed transactions may not be visible to later transactions.

MariaDB's documentation makes it difficult to tell what consistency models Galera Cluster supports. It seems likely that Galera Cluster is supposed to provide Strong Snapshot Isolation or Strong Repeatable Read, but in practice, it appears weaker than Read Uncommitted. We suggest MariaDB update the documentation to make it clear what consistency models Galera Cluster is intended to (and actually does) provi..

These results are from a brief exploration of Galera Cluster—there may be other behaviors not documented here. As always, Jepsen takes an experimental approach to safety verification: we can prove the presence of bugs, but not their absence. While we make extensive efforts to find problems, we cannot prove correctness.

4.1 Future Work

While our tests used `CONCAT` to append to strings, it seems likely that MariaDB Galera Cluster would also exhibit Lost Update with blind writes to registers, and would therefore fail the simulated banking workload used in earlier Jepsen tests—money could be destroyed or created out of thin air. We have also not explored predicates, slow networks, clock skew, or disk faults; all might prove fruitful avenues for future research.

Jepsen wishes to thank Gordan Bobic and Teemu Ollakka from the MariaDB mailing list. Our thanks to Irene Kannyo for her editorial support. This research was performed independently by Jepsen, without compensation, and conducted in accordance with the Jepsen ethics policy.