JEPSEN

Scylla 4.2-rc3

Kyle Kingsbury
2020-12-23

Scylla is a distributed database patterned after Apache Cassandra. We evaluated the community edition of Scylla 4.2-rc3, and found that both LWT and normal operations failed to meet claimed guarantees: LWT exhibited split-brain in healthy clusters, and non-LWT operations were not isolated as claimed. The split-brain issue was fixed in 4.2, and Scylla's documentation no longer claims non-LWT operations are isolated. In addition, we observed split-brain with LWT after membership changes (partially resolved), aborted reads with LWT (fixed in 4.2.1), and missing rows in response to LWT batch statements (fixed in 4.3.rc1). Scylla still exhibits split-brain, but in our testing, this was limited to membership changes concurrent with other faults. Scylla has a complementary blog post, and these findings will also be presented at Scylla Summit 2021. This work was funded by ScyllaDB, and conducted in accordance with the Jepsen ethics policy.

1 Background

Scylla is a distributed wide-column[1] database which originated as a C++ port of Cassandra, aiming for improved performance. It supports both Cassandra- and DynamoDB-compatible APIs, and is intended for high-throughput, low-latency workloads, including analytics, messaging, and other time-series data.

Scylla organizes data into *keyspaces*, which contain *tables*, which contain *rows*. Rows are uniquely identified by a primary key. Each row is physically a sorted series of $(key, value, timestamp)$ triples called *cells*, but conceptually, a Scylla row is a map of column names to values. Those values may be primitives (e.g. strings, integers, booleans, dates), counters, or collections, such as maps, lists, or sets. Collections are internally stored using multiple cells. Rows are grouped into *partitions* by a *partition key*, and those partitions are assigned via a Dynamo-style hash ring to a subset of nodes in the cluster. Each partition is replicated across multiple nodes for redundancy.

Like Cassandra, Scylla generally allows clients to write to any node at any time—even when nodes are crashed or partitioned away. Whether a write is durable depends on whether it reaches a node which can store that row; whether a write is acknowledged to the client depends on whether enough nodes respond to satisfy the client's requested consistency level. Writes are isolated, Scylla claimed, so long as they take place within a single partition:

> In an UPDATE statement, all updates within the same partition key are applied atomically and in isolation.

When there are multiple writes to a single cell, Scylla resolves them using last-write-wins (LWW): values with newer timestamps replace those with older ones. In the event of a timestamp collision, the lexicographically higher value wins.

Last-write-wins implies the potential for lost updates: if a client reads some value v_1, then writes back v_2, it is possible that a concurrent update could also observe v_1 and write v_3, *overwriting* v_2. The update of v_2 is effectively lost. To avoid this problem, Scylla users can structure their updates as *unique inserts*, taking advantage of Scylla's wide rows to store each change as a distinct column in the row. Clients can then merge those columns together on read to recover an effective value.

This approach requires that operations *commute*: writes should be able to take effect in either order. For *non-commutative* operations, Scylla (like Cassandra),

[1] There has historically been some confusion on this point. Row-oriented databases group data for each row together, whereas column-oriented databases group data for each column together. Both Cassandra and Scylla are row stores, but where Cassandra describes itself as a row store, Scylla described itself as a column store. We call both databases wide-column stores, which refers to the fact that they can store a variable number of columns per row. Scylla has updated their marketing materials to use this language as well.

offers linearizable updates via *lightweight transactions* (LWTs). LWTs allow a single Cassandra operation to proceed only if a predicate holds. They do not offer arbitrary sessions or sequences of multiple operations in a single transaction. While this prevents transactions from mixing reads and writes, a single transactional select can read multiple rows, and a single batch can insert, update, or delete multiple rows. Both of these constructs are limited to a single partition, which means that an all-LWT history over a single partition should be strict serializable.

Both Scylla and Cassandra use a Paxos-based consensus algorithm for these transactions, but where Cassandra requires four round trips per transaction, Scylla requires only three. Other consensus algorithms, such as Raft, can achieve consensus in one round trip, which is why Scylla is laying the groundwork for a Raft-based LWT implementation.

2 Test Design

Scylla had an existing Jepsen test suite adapted from Cassandra's Jepsen tests. We reviewed and significantly expanded this test suite for Scylla 4.2-rc3 through 4.2.rc5, including the creation of new, more aggressive workloads and more sophisticated nemeses for fault injection. Our new tests ran on clusters of five Debian Buster nodes, deployed in LXC, Docker, and EC2.

We made several tuning changes to Scylla's default configuration to speed up testing. By default, Scylla takes over a minute to detect a node failure, and 300 seconds to recover from a process crash, due to a temporary deadlock involved in gossip on boot. We lowered `phi_convict_threshold`, `ring_delay_ms`, `shadow_round_ms`, and adjusted other settings to reduce startup and recovery times.

During these tests, we injected a variety of faults, including network partitions, process kills, process pauses, clock skew, and membership changes, including adding, repairing, decommissioning, and forcibly removing nodes. In addition, we measured behavior both with and without custom timestamp generators which introduced synthetic clock skew and increased the probability of timestamp collisions.

2.1 Workloads

Scylla's original test suite included workloads for CQL (Cassandra Query Language) maps and sets, both of which insert several elements with consistency `ONE` into a single collection, and attempt to read them back with a final read at consistency `ALL`. The batch workload inserts pairs of rows together, and attempts to read back both rows. A counter workload creates a single CQL counter and attempts to increment it repeatedly. Reads verify that the counter value remains within expected ranges. A dedicated materialized-view workload updates map values and queries a materialized view to see if those changes are reflected.

While we briefly evaluated these workloads, the present work focused on Scylla's lightweight transaction safety. To verify LWT safety, we used three workloads.

The first, cas-register, uses LWT to perform reads, writes, and compare-and-set operations over several rows. It uses Knossos to verify that the history of operations over each individual row is linearizable. Knossos is exponential with respect to concurrency, and concurrency (thanks to indeterminate responses) rises rapidly in Jepsen testing. This limits the length of histories to a few hundred operations per row.

To complement the Knossos checker, we designed list-append and wr-register workloads, both of which use Elle to search for violations of strict serializability. Elle uses knowledge of the history and data structures involved to infer constraints on the order of versions of each individual key, and the order of transactions over those keys. Cycles in these constraint graphs correspond to isolation anomalies. The list-append test performs transactions which append unique integers to CQL lists, and reads those lists by primary key, whereas the wr-register test writes unique integers to individual rows, rather than CQL collections.

Both workloads perform LWT transactions composed of a single `SELECT` or `BATCH` update. Scylla prohibits mixing reads and writes in a single query, as well as queries which read multiple rows with CQL collections, and LWT queries that cross partition boundaries. Even with these restrictions we are able to verify single key linearizability, as well as limited multi-key strict serializability within a single partition.

We also designed a variety of special-purpose workloads to investigate anomalous behavior. Batch-return examines the rows returned in response to LWT batches. This test verifies that returned rows correspond to requested updates in each batch. Write-isolation performs non-LWT writes to multiple cells, and performs concurrent reads, looking for cases where only some values came from the same write: evidence of read skew.

3 Results

Our testing focused on lightweight transaction safety, but along the way we uncovered some additional behaviors in non-LWT operations. We'll start by discussing some minor issues around timestamps, batch returns, and non-LWT isolation, then cover stale reads and split-brain in lightweight transactions.

3.1 Destructive INSERTs

We uncovered an unusual behavior with Scylla's existing tests for CQL sets and maps: when we tested with imperfect timestamps, they appeared to lose acknowledged inserts. Higher degrees of clock skew resulted in more writes lost, but even skews as small as one second resulted in lost updates. This behavior was particularly surprising because distinct set and map inserts should commute. In other words, adding `a` then `b` to a set ought to be the same as adding `b` then `a`.

This behavior turned out *not* to be a bug; it is, in fact, documented behavior. To understand why, we have to look at the transactions performed during the set (or map) workload. They begin by creating a table with a set (map) column, and inserting a single row:

```
CREATE TABLE sets (
  id    int PRIMARY KEY,
  value set<int>
);
INSERT INTO sets (id, value)
  VALUES (0, {});
```

After creating this empty set, clients perform updates: each adding a unique element to the set. For example:

```
UPDATE sets SET
  value = value + {1} WHERE id = 0;
```

These `UPDATE` statements all commute with one another, but the `INSERT` and `UPDATE` do not. If the `INSERT` receives a higher timestamp than an `UPDATE` it will silently negate that update's writes—regardless of the real-time order. This is surprising for three reasons.

First, database users often assume linearizability implicitly: if the `INSERT` completes, an `UPDATE` begun after that completion should take effect later. This is true for LWT, but not for normal Scylla operations. Users accustomed to Scylla (or other Cassandra-style databases) are likely aware of this behavior, and instead might ask what *consistency level* was involved in these operations, since they could have taken place on disjoint nodes. This is a red herring: the behavior is a consequence of last-write-wins timestamp arbitration, and choosing consistency level `QUORUM` or `ALL` does nothing to prevent it.

Second, `INSERT` and `UPDATE` in CQL have different semantics than in most query languages. In SQL for example, `INSERT` creates a new row and `UPDATE` alters an existing row. A successful `INSERT` and `UPDATE` pair in SQL can only execute in one order: the `INSERT` *must* have taken place before the `UPDATE`, because otherwise the `UPDATE` would have had no row to modify. By contrast, CQL's `INSERT` and `UPDATE` are (almost) indistinguishable: both mean "upsert". `UPDATE` statements create new rows when none exist, and `INSERT` statements can succeed even when every replica already has data for the row being "inserted"; it overwrites any cells with a lower timestamp.

Third, users accustomed to working with CRDTs might expect that CQL sets are something like an OR-set or G-set: sets which can always safely add elements but where concurrent removals might be dropped. Inserting a value of `{}` would be safe in these cases: one might expect the insert to be a no-op (essentially, an addition of no elements), or to delete causally prior values, but not to delete causally concurrent or later values. Scylla, like Cassandra, does not do either of these things. Insert is a destructive operation by design. In fact, writing any collection literal (e.g. `{}` or `(1, 2)`) is internally implemented by writing a deletion tombstone followed by the new values.

This behavior is somewhat documented. For example, the insert documentation states:

> Note that unlike in SQL, INSERT does not check the prior existence of the row by default: the row is created if none existed before, and updated otherwise. Furthermore, there is no means to know which of creation or update happened.

Likewise for `UPDATE`. While the Cassandra set documentation shows an `INSERT` followed by an `UPDATE` as if the two should happen in order, and the Scylla datatypes documentation does too, neither of these examples explicitly claims that that order is guaranteed, rather than likely. In general, operations in Scylla and Cassandra should be expected to (occasionally) take place in arbitrary orders. Similarly, the names of `INSERT` and `UPDATE` are suggestive but not definitive: since `UPDATE ... SET value = {}` can destroy information, and `UPDATE` and `INSERT` are effectively the same operation, `INSERT` can destroy information too.

If you were surprised by this, you're not alone. The Cassandra engineers who originally designed this test didn't realize `INSERT ... {}` was unsafe. This work-

load was ported to Scylla by Scylla engineers, reviewed by Jepsen, and reviewed again by multiple Scylla engineers before one realized the mistake. Jepsen posted an informal survey which asked CQL users what they'd expect to happen in this scenario, and out of eleven responses, no one correctly predicted this outcome.[2]

Users may be able to work around this by only performing (commutative) `UPDATE` operations, without initial `INSERT`s. Scylla's documentation no longer claims that non-LWT operations are isolated, explains timestamp conflict behavior, and mentions that inserts of empty maps are the same as deletions.

3.2 Aborted LWT Reads

Infrequently, under network partitions and process crashes, LWT writes to Scylla could appear to fail, but actually succeed. In particular, the error message `UnavailableException: Not enough replicas available for query at consistency QUORUM` should denote the operation definitely did not take place, but those operations may in fact be visible to later reads. For example, in this list-append test, a failed append of 52 to key 618 was observed by a later read:

```
759 :fail :txn [[:append 629 127]
                [:append 618 52]]
...
648 :ok   :txn [[:r 618 [50 52 69 74]]]
```

If we take this exception to mean the append of 52 did not commit, then this pair of operations constitutes an aborted read! But how *should* we interpret this error?

As of September 29, 2020, ScyllaDB's documentation did not appear to include any description of what error messages mean, or whether their results were definite. Datastax's Java client documentation describes this error as an "[e]xception thrown when the coordinator knows there is not enough replicas alive to perform a query with the requested consistency level," which the Cassandra error docs confirm. The Cassandra diagram's documentation shows that a coordinator returns an `UnavailableException` when *no* communication with replicas has taken place—whereas other exceptions, like `WriteTimeout`, are thrown when a coordinator *has* issued requests to a replica.

Users might reasonably conclude that an `UnavailableException` denotes a definite failure. This is not the case: Scylla (unlike Cassandra) checks availability multiple times during the LWT process. It can therefore throw `UnavailableException` in scenarios where requests have already been issued. Scylla 4.3.rc1 addresses this issue by returning `WriteTimeout`.

3.3 Weird Return Values From Batch Updates #7113

Many query languages include some notion of a *batch transaction*: a statement which executes multiple substatements together, and returns the results of their application. In Scylla, we might perform an LWT `BATCH` statement like so:

```
BEGIN BATCH
UPDATE batch_ret SET a = 3
  WHERE key = 1 IF lwt_trivial = null;
UPDATE batch_ret SET b = 5
  WHERE key = 2 IF lwt_trivial = null;
APPLY BATCH;
```

The `IF` condition signifies that these updates should take place using LWT. Conditionals are mandatory in CQL; we use `lwt_trivial` (a column defined in our schema, but whose value is always null) to allow these updates to always succeed.

Individual LWT `UPDATE` statements return a single row with an `[applied]` field, as well as the prior value for that row's key and any fields used in the LWT conditional. The return value of batch was undocumented, but one might expect it to be a series of rows corresponding to the results of each statement in the batch. Indeed, this is sometimes the case:

```
({:[applied] true, :key 1, :lwt_trivial nil}
 {:[applied] true, :key 2, :lwt_trivial nil})
```

… but sometimes not:

```
({:[applied] true, :key 2, :lwt_trivial nil}
 {:[applied] true, :key 1, :lwt_trivial nil})
```

In practice, `UPDATE`'s return values were ordered by clustering key, rather than the order they were written in the `BATCH` statement. This, combined with the fact that `update` returns the prior values of LWT keys (which may have been null!), meant that it was (in general) impossible to figure out which returned row corresponded to which `UPDATE` statement. Two updates could return a single row, like so:

```
({:[applied] true,
  :key null,
  :lwt_trivial nil})
```

[2] After asking respondents what they thought might happen, we explained the actual behavior and asked users how they felt. Responses included "Surprising", "That feels broken", "[N]ot what I expected", "[J]ust wrong", "This looks quite unsafe", and "I feel bad".

The Scylla team confirmed this was expected behavior. However, we also observed rows which were missing altogether. Here, a `BATCH` which updated keys 1 and 2 returned a result set without any value for key 2:

```
{:[applied] true, :key 1, :lwt_trivial nil})
```

This was in fact a bug, caused by Scylla sometimes (but not always!) stripping out result rows which had a `nil` prior key. Scylla resolved the problem by returning batch results in statement order, which allows clients to predictably identify which result corresponds to which update, and has documented the behavior.

3.4 Normal Writes Are Not Isolated

Scylla's DML documentation made repeated claims that INSERT, UPDATE, DELETE, and BATCH are all isolated (at least, when limited to a single partition). This is not the case: clients which e.g. only add elements to a single CQL set could observe states like `{1}` and `{2}`. Such a history cannot be understood to be isolated, in the usual sense, because there is no total order of operations which could result in both values. The existence of the state `{1}` implies that the addition of 2 must have followed 1, but the existence of `{2}` implies that the addition of 1 must have followed 2. Non-LWT updates to collections and counters are fundamentally concurrent.

This problem is not limited to partial updates—writes which completely replace the value of some column are not isolated either. We repeatedly observed isolation violations both in batch and single-row updates. In the `write-isolation` test, we perform write operations which set every value in a group of keys to either `+x` or `-x`. Any read of that group should see that every key has the same absolute value. Instead, we observed transactions like:

```
[[:r 4 -5] [:r 3 -2] [:r 5 -3]]
```

Here, key 4's value is -5, key 3's value is -2, and key 5's value is -3: values from three completely separate writes have been jumbled together. This problem occurs in healthy clusters, even with consistency level `ALL` for reads and writes, and when using the standard Scylla client's `AtomicMonotonicTimestampGenerator`. At a thousand operations per second (evenly split between reads and writes), we observed isolation violations roughly every 20 seconds. By quantizing timestamps, we could induce anomalies in just a handful of writes.

The author first reported this problem with Cassandra in 2013. In 2014, Cassandra realized that their read-repair mechanism could *also* violate partition-level isolation, and decided not to address the problem at that time. Scylla's engineers reported these problems again, including additional cases where Cassandra could fail to meet its claimed isolation guarantees, in 2017. As of September 2020, providing row-level isolation in the face of timestamp collision remains an open issue in Cassandra, the documentation ticket is unaddressed, and Cassandra's documentation still insists that writes are "performed with full row-level isolation." This problem continues to vex users, who occasionally discover this behavior when it results in logical data corruption.

Some engineers have argued that this behavior is still isolated: it's simply that writes in systems like Scylla and Cassandra don't mean what most people think of as a write. If a write is understood to mean "maybe set the value for this cell, depending on whether and how other people have already or will, at some future time, write to it" then this behavior can indeed be isolated. It's just that Scylla is exercising its freedom to do whatever it wants with one's writes. The problem, of course, is that if one actually wants to *definitely* write a value, there is no timestamp one can choose which cannot be ruined by some other client. Every client must carefully coordinate their timestamp choices. Avoiding the need for careful client coordination is precisely why one wants an isolation property in the first place! Whether users are generally aware of this nondeterministic interpretation, and whether it is possible to write the kinds of applications which users want to write using "row-level isolation", remain under debate.

Scylla is already aware of these issues—tickets like 2379 discuss various cases where partition-level isolation fail to hold. However, Scylla's documentation still contained strong isolation claims. We have opened an issue to resolve the documentation error.

3.5 LWT Stale Reads

In both list-append and wr-register tests, we observed repeated violations of single-cell linearizability in workloads which should have been strict serializable. Reads could observe values which had been replaced tens of seconds ago. These problems were exacerbated by network partitions, but also occurred in healthy clusters with no exogenous faults.

For example, this test run set key 95 to 5, and after that write completed, set key 95 to 8. Reads confirmed the value was 8, and clients went on to set key 95 to 9, 12, and eventually 1824. After forty seconds of new values being written and read, a read of key 95 abruptly returned 5 again: a stale read!

Similarly, list-append tests reliably observed cycles where the real-time order of transactions was incompatible with the actual contents of reads and writes. For example, this test run contained the following real-time transactional anomaly:

```
G-single-realtime #0
Let:
  T1 = {:type :ok, :f :txn, :value [[:r 43 [25 26 27]]], :time 109236930848,
        :process 1673, :index 5026}
  T2 = {:type :ok, :f :txn, :value [[:append 43 4]], :time 108768382054,
        :process 1632, :index 4988}

Then:
  - T1 < T2, because T1 did not observe T2's append of 4 to 43.
  - However, T2 < T1, because T2 completed at index 4988, 0.196 seconds
    before the invocation of T1, at index 5005: a contradiction!
```

Both of these transactions contain only a single read or write—they map to a single `SELECT` at consistency level `SERIAL`, and an LWT `UPDATE`, respectively. This is a minimal example of a much larger problem: of the 1172 acknowledged transactions performed in this two-minute test, 264 of them were involved in a single, sprawling linearizability violation.

This turned out to be a symptom of a more fundamental problem: LWT split-brain.

3.6 LWT Split-Brain

In addition to stale reads, list-append tests contained reads of lists which could not have arisen from any sequence of appends—despite all operations using `SERIAL` reads and LWT writes. This behavior occurred in healthy clusters without any faults. For example, take this history, where reads of key 36 returned the following states:

```
[17 18 19]
[17 18 19 20]
[17 18 19 20 22]
...
[17 18 19 20 22 23 24 5 9 11 1 26 27 21]
[22 23 24]
[17 18 19 20 22 23 24 5 9 11 1 26 27 21]
```

After 4.6 seconds of stable existence, the entire list momentarily disappeared, leaving only 22, 23, and 24. Meanwhile, key 39 exhibited what appears to be a classic case of split-brain. Reads of key 39 observed:

```
[2]
[2 6 1 4 5 3]
[2 12]
[2 12]
[2 12]
[2 6 1 4 5 3 12 16 14 29]
[2 6 1 4 5 3 12 16 14 29 31]
```

For roughly four seconds, node n4 thought the value was `[2 12]` while node n5 repeatedly observed `[2 6 1 4 5 3]`. Afterwards, the append of 12 appeared in the middle of n5's version.

Scylla traced this issue to an extant bug first discovered in 2019. When calculating the hash of a row, Scylla would inadvertently stop as soon as it encountered a null column. Since Jepsen's tests involved a null `lwt_trivial` column, changes to the `value` column did not affect row hashes. This allowed rows with completely different values to be perceived as identical during lightweight transactions.

Scylla's engineers corrected the hash calculation in Scylla 4.2.

3.7 LWT Split-Brain With Membership Changes

Subsequent testing with membership changes revealed that when nodes are removed and added from the cluster, LWT operations could exhibit unusual split-brain behavior. Clients can read and write to what appear to be separately-evolving versions of the same record. For example, in this test run, key 391 alternated between two completely independent timelines. Process 48 observed versions `[3 9 10 ...]` while process 44 observed `[15 16 17 24 ...]`.

```
48 :invoke  :txn [[:r 391 nil]]
48 :ok      :txn [[:r 391 [3]]]
44 :invoke  :txn [[:r 391 nil]]
44 :ok      :txn [[:r 391 [15 16 17]]]
48 :invoke  :txn [[:r 391 nil]]
48 :ok      :txn [[:r 391 [3 9 10 27 28]]]
44 :invoke  :txn [[:r 391 nil]]
44 :ok      :txn [[:r 391 [15 16 17 24 25]]]
```

Even stranger, a single write could be applied to both timelines. Here, an append of 9 and 10 to key 1897 appeared to take effect only on one fork, but a subsequent append of 11, 12, and 13 was visible on *both*:

```
729     :invoke :txn    [[:append 1896 20] [:append 1897 9] [:append 1897 10]]
729     :ok     :txn    [[:append 1896 20] [:append 1897 9] [:append 1897 10]]
1072    :invoke :txn    [[:append 1896 21] [:append 1897 11] [:append 1897 12]
                         [:append 1897 13]]
1072    :ok     :txn    [[:append 1896 21] [:append 1897 11] [:append 1897 12]
                         [:append 1897 13]]
...
877     :invoke :txn    [[:r 1897 nil]]
877     :ok     :txn    [[:r 1897 [11 12 13]]]
1017    :invoke :txn    [[:r 1897 nil]]
1017    :ok     :txn    [[:r 1897 [11 12 13]]]
729     :invoke :txn    [[:r 1897 nil]]
670     :invoke :txn    [[:r 1897 nil]]
729     :ok     :txn    [[:r 1897 [9 10 11 12 13 18]]]
670     :ok     :txn    [[:r 1897 [9 10 11 12 13 18]]]
```

Scylla believes this issue has three causes.

First, repair-based streaming could stream data from only one or two nodes, rather than a majority of replicas. If a node was stopped or partitioned away, the recovering node could fail to observe committed writes, causing data loss or logical state corruption.

Second, in CQL, LWT list append operations choose a UUID for each list element based on the local system clock, rather than using the LWT timestamp. These timestamps could conflict across multiple nodes, causing elements to be lost or reordered.

Finally, ScyllaDB allowed cluster membership changes to execute concurrently: nodes could be added or removed while (e.g.) a replace-node operation was in progress, but Scylla's membership system assumed that changes occurred sequentially. Scylla asserts that executing concurrent membership changes or removing a node which might still be running constitute operator errors. Users must ensure a node is truly dead before issuing a `nodetool remove` command. These hazards were undocumented.

Scylla reports that they have patches for the first two issues. However, Jepsen tests continue to observe split-brain with node removal and network partitions. Scylla believes this is due to the test removing nodes which the cluster believed were down but were not totally, permanently dead. Cursory testing where nodes are killed prior to removal appears to confirm this hypothesis.

№	Summary	Event Required	Fixed In
7258	Aborted reads of failed "not enough replicas" LWT ops	Partitions, crashes	4.1.9, 4.2.1
7113	Missing values from BATCH updates	None	4.3.rc1
7170	Single-partition updates aren't isolated	None	Documented
7116	Split-brain with LWT	None	4.2
7359	Split-brain with LWT due to improper repair streaming	Membership & crash	829b4c1
7611	Split-brain with LWT due to timestamp conflict	Membership	Unresolved
7351	Split-brain with LWT due to concurrent membership	Membership & partition	Unresolved

4 Discussion

We found seven issues in Scylla 4.2-rc3. LWT updates could return `UnavailableException` for writes which actually committed. Batch updates could fail to return results for freshly inserted rows. Non-LWT updates claimed to be isolated, but were not. With LWT, healthy clusters exhibited stale reads and split-brain scenarios in which values fluctuated between multiple apparent timelines. Membership changes could also induce split-brain behavior in LWT operations for a variety of reasons.

Aborted reads, batch updates and LWT split-brain in healthy clusters have been fixed in 4.3.rc1. Some issues with split-brain with membership changes have been resolved in recent development builds, but others remain; they may be resolved as Scylla moves to a Raft-based membership system.

In addition, Scylla exhibits normal Cassandra behaviors around last-write-wins conflict resolution. This may be surprising. As we discovered, even experienced database engineers can fail to anticipate the behavior of simple tests! Inserts may destroy data now or in the future. Read-modify-write can result in lost updates. Cells written together may not be visible together. These issues are (to varying degrees) documented in both Scylla and Cassandra, and can be mitigated, as we discuss below. Scylla has updated the DML documentation: it no longer claims that non-LWT operations are isolated, and explains in more detail what can go wrong when timestamps conflict.

Cassandra counters and materialized views are known to be unsafe and Scylla mirrors this behavior. Users should expect counters to be approximations. Materialized views may fail to reflect updates.

As of December 11, 2020, Scylla's development builds appear close to offering strict serializability for the limited (i.e. single read or batch write, both within a single partition) transactions expressible under LWT. Cross-partition operations are not isolated, but we expect partitions to be independently linearizable. The only conditions under which we presently observe split-brain involve concurrent membership changes, or membership changes combined with other faults.

As always, we note that Jepsen takes an experimental approach to safety verification: we can prove the presence of bugs, but not their absence. While we try hard to find problems, we cannot prove the correctness of any distributed system.

4.1 Recommendations

In Scylla 4.2-rc3, users should be aware that LWT operations, even in healthy clusters, may be nonlinearizable. Reads may return stale data or inconsistent views of a record's history. Lists and sets can have some or all of their values disappear then reappear. As a workaround, users may be able to restructure their schemas and/or queries such that LWT-conditional fields occur *last* in a row; e.g. by naming those fields `zzz_foo`, rather than `foo`. This problem is fixed in Scylla 4.2; we recommend that users upgrade to at least this version.

Membership changes in 4.2-rc3 through 4.2-rc5 appeared risky: we were able to induce split-brain behavior, even with LWT, by decommissioning, wiping, and adding nodes. This behavior was exacerbated by partitions, process crashes, and process pauses, but we don't yet know the details. It may be worth temporarily pausing safety-critical operations during and shortly after a membership change. Recent development builds have reduced, but not eliminated, these risks.

The Scylla cluster management procedures documentation informs users that they can remove a dead node by checking that its status is listed as `DN` in `nodetool status`, then issuing a `nodetool remove` command. This is unsafe: running nodes can be perceived as down by any number of other Scylla nodes, e.g. due to a network partition, IO hiccup, or VM migration. Removing a node under these conditions can result in split-brain, causing lost updates and logical data corruption. Users must confirm a node is *truly* dead before removing it. Scylla recommends that users log in to the down node to shut down the Scylla process, physically unpower the machine, or terminate the VM or container it runs in. If these procedures cannot be completed, the node cannot be safely removed.

Likewise, users must take care not to issue a membership operation before all previous membership operations have completed. It is unclear how to tell when a membership operation has completed: there is no method to block on a previously submitted membership change, and in our testing, `nodetool status` often differeed from node to node. Scylla states that users must wait for unanimous agreement on cluster state before proceeding; cluster changes cannot be safely performed when some nodes are unreachable.

Scylla maintains that these membership operations are illegal: users should have known not to try them. However, these rules remain completely undocumented. It is reasonable to expect that users *will* attempt to remove unresponsive nodes, or issue multiple membership changes to a struggling cluster—and if they do, they could encounter split-brain. We recommended that Scylla clearly explain these constraints in the cluster management documentation.

Users may have designed applications relying on Scylla's claims that writes to a single partition are isolated and atomic. These claims are inaccurate. We suggest reviewing any operations which update multiple cells which depend on one another. For example, setting a row's `password-hash` and `salt` in a single `UPDATE` does not necessarily mean that the two fields will match, which could leave the user unable to log in. Consider using LWT for these writes.

When safety is critical, we recommend that users employ LWT whenever updating existing data. Without LWT, avoid inserting initial values which will be updated later. Instead, rely on the fact that CQL updates work well on nonexistent cells: one can add elements to a CQL map or set without the field existing beforehand. This prevents updates from being lost—so long as one limits oneself to a commutative subset of CQL operations. Where possible, structure applications so that they write to any given cell exactly once. Employing Flake IDs for both timestamps and key construction might help prevent conflicts.

These risks can be mitigated by maintaining (and alerting on!) closely synchronized clocks for all Scylla nodes *and* their clients, and by reducing the frequency of updates. So long as updates are infrequent compared to the scale of clock errors, these problems are unlikely to occur. This does not mean they are impossible: as Cassandra users continue to observe, normal NTP error, misconfigurations, memory errors, and unsolved mysteries can lead to data written with timestamps anywhere from milliseconds to thousands of years in the future.

4.2 Future Work

Jepsen has not evaluated Scylla's behavior with respect to schema changes, and our membership-change testing is still in the early stages. Nor have we investigated Scylla's DynamoDB-compatible API, Alternator. Scylla has done extensive testing with filesystem-level fault injection—we would like to apply these with Jepsen as well.

Scylla plans to address the outstanding issues we found, but membership changes will likely remain problematic for some time. Scylla's membership protocol is based on Cassandra's gossip system rather than a consensus system, which makes it difficult to ensure changes occur sequentially and that nodes have a consistent view of the cluster state. Even though LWT operations go through Paxos, nodes may disagree about the quorum required for that Paxos operation. Once Scylla finishes rewriting their membership system to use Raft, we suspect these problems will be easier to solve.

This work was funded by ScyllaDB, and conducted in accordance with the Jepsen ethics policy. Jepsen wishes to thank the entire Scylla team—in particular, Kamil Braun, Peter Corless, Piotr Jastrzębski, Dor Laor, Duarte Nunes, Konstantin Osipov, Alejo Sánchez, and Pavel Solodovnikov. We would also like to thank Irene Kannyo for her editorial support during preparation of this manuscript.