H2 database and MVCC - 3rd week

Sungkeun Kim

Report History

Week 1	Understanding MVCC and test program
Week 2	Issue analysis - Isolation level with MVCC(#174, #216) - Part 1(Pending)
Week 3	Issue analysis - executing UPDATE and SELECT FOR UPDATE simultaneously

1. Overview of weekly report

Previous week, I reported the issue #174 which is about Isolation level under the MVCC. Currently, H2 database doesn't support for changing the lock mode(LOCK_MODE), and implementation of isolation level in MVCC is not suitable for me because it needs a deep understanding of the whole h2 database system.

In this week, I chose another issue which is obvious bug. I found a open issue related to the MVCC which is the #180. The issue #180 claims that when "UPDATE" and "SELECT FOR UPDATE" queries are executed simultaneously, some rows are lost from the database. I reproduced this issue with a test program and narrowed down the scope of the debugging areas. Currently, because of lack of the deep understanding of the behavior of the h2 database, I am working on reviewing the h2database source codes.

For the next week, I will keep working on analysis of the h2 source codes for understanding the details and solve the issue #180.

2. Reproduction of the issue

2.1 Precondition

Test version	1.4.194
MVCC	enabled
Persistent / In-memory	In-memory

2.2 Reproduction Scenario

The problem occurs when both “UPDATE” and “SELECT FOR UPDATE” queries are executed simultaneously. Below is the simplified scenario. (Test program link)

Main Thread

```
CREATE TABLE test (
      id PRIMARY KEY,
      lastUpdated TIMESTAMP NOT NULL
)

INSERT INTO test (id, lastUpdated) VALUES (1, 2017-04-13)
INSERT INTO test (id, lastUpdated) VALUES (2, 2017-04-13)
INSERT INTO test (id, lastUpdated) VALUES (3, 2017-04-13)
```

Thread 1

```
while (true) {

  UPDATE test SET lastUpdated = 2017-04-13
  COMMIT;

}
```

Thread 1

```
while (true) {

  SELECT * FROM test WHERE id = 1 FOR UPDATE
  COMMIT;
  // Sometimes, there is no row with id "1"

}
```

<Figure 1. Reproduction scenario>

2.3 Narrowing down the scope of the debugging area.

Before I start finding the problem, I have tested same scenario under different condition. This will be helpful for me to review the source codes.

1. MVCC vs Non-MVCC

Non-MVCC	**No BUG**
MVCC	BUG

2. In-Memory vs Non-In-Memory

In-Memory (jdbc:h2:mem:test;)	BUG
Non In-Memory **(jdbc:h2:tcp://localhost/~/test;)**	**BUG**

Interesting thing here is that I could check whether lost rows actually disappeared from the original db by using persistent mode. This is a real problem!

3. "SELECT FOR UPDATE" vs "SELECT"

```
Thread 1

while (true) {

  UPDATE test SET lastUpdated = 2017-04-13
  COMMIT;

}
```

```
Thread 1

while (true) {

  SELECT * FROM test WHERE id = 1 //FOR UPDATE
  COMMIT;
  // Bug did not occur!
}
```

In this case, I found "SELECT FOR UPDATE" query is the key factor of the problem.

4. “Update every row” vs. “Update several rows

```
                    Thread 1
while (true) {

  UPDATE test SET lastUpdated = 2017-04-13
WHERE ID in (1, 2)
  COMMIT;

}
```

```
                    Thread 1
while (true) {

  SELECT * FROM test WHERE id = 1 FOR UPDATE
  COMMIT;
  // Bug did not occur!

}
```

In this case, I couldn’t reproduced the problem with the query that updates few rows. However, It does not guarantee the problem never be happened in this query. I just found that update every row increases the probability of the problem.

In summary, preconditions that 1. SELECT FOR UPDATE and 2. UPDATE every row under 3. MVCC mode increases the occurrence probability of this bug. Also, the fact that this bug occurs under the non-memory mode makes the problem more serious.

3. Source code analysis

I have started to analysis the source code in earnest since this week.

3.1. Where is the location of saving MVCC option?

In order to keep track of the MVCC, we need to know the where MVCC option is saved.

```
// org/h2/server/TcpServerThread.java
public void run() {
    ....
    int len = transfer.readInt();
    for (int i = 0; i < len; i++) {
        ci.setProperty(transfer.readString(), transfer.readString());  // MVCC=TRUE;
     }
```

Every connection options are store in class ConnectionInfo when the Tcp Server is initializing. And then, the option is transferred into the class Database.

```
// org/h2/engine/Database.java
public Database() {
    ....
    this.multiVersion = ci.getProperty("MVCC", dbSettings.mvStore);
}

/**
 * Check if multi version concurrency is enabled for this database.
 *
 * @return true if it is enabled
 */
public boolean isMultiVersion() {
    return multiVersion;
}

// call stack
java.lang.Exception
        at org.h2.engine.Database.<init>(Database.java:244)
        at org.h2.engine.Engine.openSession(Engine.java:60)
        at org.h2.engine.Engine.openSession(Engine.java:167)
        at org.h2.engine.Engine.createSessionAndValidate(Engine.java:145)
        at org.h2.engine.Engine.createSession(Engine.java:128)
        at org.h2.server.TcpServerThread.run(TcpServerThread.java:149)
        at java.lang.Thread.run(Thread.java:745)
```

3.2 Where is the location of handling db query?

DB queries are processed in the class Session. Session class has a Database class so that each session manipulates the database separately.

```
// org/h2/engine/Session.java

Class Session extends SessionWithState {

 private final Database database;
 private final ArrayList<Table> locks = New.arrayList();

 /**
  * Commit the current transaction. If the statement was not a data
  * definition statement, and if there are temporary tables that should be
  * dropped or truncated at commit, this is done as well.
  *
  * @param ddl if the statement was a data definition statement
  */
  public void commit(boolean ddl) {
      // ...
      if (locks.size() > 0) {
          for (int i = 0, size = locks.size(); i < size; i++) {
            Table t = locks.get(i);
            if (t instanceof MVTable) {
                ((MVTable) t).commit();
            }
      }
      // ...
      if (database.isMultiVersion()) {
      / …
  }


// Call Stack

/*SQL l:49 #:1*/SELECT * FROM test WHERE entity_id = ? FOR UPDATE {1: '3'};
at org.h2.engine.Session.commit(Session.java:478)
at org.h2.command.dml.TransactionCommand.update(TransactionCommand.java:46)
at org.h2.command.CommandContainer.update(CommandContainer.java:78)
at org.h2.command.Command.executeUpdate(Command.java:254)
at org.h2.server.TcpServerThread.process(TcpServerThread.java:345)
at org.h2.server.TcpServerThread.run(TcpServerThread.java:159)
at java.lang.Thread.run(Thread.java:745)
```

References

[1] H2 - Advanced

http://www.h2database.com/html/advanced.html#mvcc

[2] H2 Github-issue

https://github.com/h2database/h2database/issues/180

[3] Github of Test programs

https://github.com/danguria/H2Exercise/commit/2ec060fcc67b621960fb170f9f642726126d1b60