Bugs found in Database Management Systems

Keqiang Li

School of Data Science & Engineering, East China Normal University, Shanghai, China

This page lists all bug reports that we created for widely-used Database Management Systems (DBMS), such as MySQL, PostgreSQL, and TiDB. We are thankful to the DBMS developers for responding to our bug reports and fixing all the bugs that we found.

Unique fixed bugs

TiDB

Dirty write of SI

Data Found:

2019/12/12

Severity:

(S1)Critical

Test Case:

Transaction ID	**Operation Start Timestamp**	**Operation End Timestamp**	**Operation Detail**	**State**
739	104865095693539	104865097351053	Update table_7_2 set attribute1=-5012153 and attribute2=2240641.4 where primarykey=676	Success
723	104865111029861	104865114503063	Update table_7_2 set attribute1=852150 where primarykey=676	Success
739	104865115133146	104865118426143	Commit	Success

Transaction 739 writes a record(676) of table_ 7_2. Transaction 723 also writes this record before transaction 739was committed, resulting in dirty write.

Read inconsistency of SI

Data Found:

2019/12/14

Severity:

(S1)Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
904	105092107947136	105092115337912	Update table_8_2 set attribute6=-0.386 where primarykey=3873	Success
904	105092144188546	105092148615512	Commit	Success
No other transaction writes the record(3873) of table_8_2 from 105092144188546 to 105092186223727				
907	105092111994965	105092150997653	Update table_8_2 set attribute6=0.484 where primarykey=3873	Success
907	105092182650339	105092186223727	Commit	Success
No other transaction writes the record(3873) of table_8_2 from 105092182650339 to 105092189561012				
914	105092187673511	105092189561012	Select attribute1 from table_8_2 where primarykey=3873	Success
914	105092189611217	105092191263618	Select attribute6 from table_8_2 where primarykey=3873	Success(attribute6=-0.368)

In the table above, for the record(3873) of table_ 8_2, there are two historical versions of attribute6, the first is -0.386, the creation time is (105092144188546105092148615512); the second is 0.484, the creation time is (105092182650391050921862237 27). The start timestamp of transaction 914 is in the interval (105092187673510509219126361 8), then the attribute6 of transaction 914 reading record 3873 should be 0.484, but TiDB returns -0.386, indicating that there is a problem with the consistency reading of TiDB.

Schema version check error

Data Found:

2020/01/01

Severity:

(S2)Serious

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
712	107685099231231	107685110245123	Drop db0.table_1_2	Success
723	107685095692321	107685097353242	Select * from db1.table_5_1 where primarykey=2114	Success
No other transaction modify schema of db1				
723	107685111022412	107685114502321	Update db1.table_5_1 set attribute2=8132130 where primarykey=6123	Exception(Information schema is changed)

The first line modifies db0's schema information, and the fourth line modifies db1's data with exception: information schema is changed, which is a bug.

Timestamp acquisition mechanism error of RC

Data Found:

2020/03/01

Severity:

(S1)Critical

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
232	112242421212321	112242421786874	Select * from table_2_1 where primarykey=4323	Stall(never response)

Under the RC isolation level recently developed by TiDB team, in order to optimize the performance of timestamp acquisition, asynchronous timestamp acquisition mechanism is adopted, but there are internal problems in this mechanism, as shown in the above table.

Unique confirmed bugs

TiDB

Update BLOB data error

Data Found:

2020/05/02

Severity:

(S3)Non-Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
1	2020-08-05 15:52:27.477	2020-08-05 15:52:27.484	Update tablecsacas0 set attributeqwdcwq3=FILE("./data_case/obj/12obj_file.obj") where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	Success
1	2020-08-05 15:52:27.495	2020-08-05 15:52:27.501	Update tablecsacas0 set attributeqwdcwq3=FILE("./data_case/obj/12obj_file.obj") and other column where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	Success
1	2020-08-05 15:52:27.508	2020-08-05 15:52:27.512	Select attributeqwdcwq3 and other column from tablecsacas0 where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 for update	Success and Return attributeqwdcwq3 = NULL (ERROE)

For BLOB data type, when the new value and the old value written by the update operation are for the same binary file, the value actually written is null and success is returned.

The long lock of the FOR UPDATE statement and the long lock of the UPDATE statement are not mutually exclusive

Data Found:

2020/05/25

Severity:

(S1)Critical

Test Case:

```
drop database if exists db1;
create database db1;
use db1;
create table t1(a int primary key, b int);
create table t2(a int primary key, b int, constraint fk1 foreign key(b) references t1(a));
create view view0(t2_a,t2_b,t1_b) as select t2.a,t2.b,t1.b from t2,t1 where t2.b=t1.a;

insert into t1 values(1,2);
insert into t1 values(2,3);
insert into t1 values(3,4);
insert into t1 values(4,5);
insert into t1 values(5,6);

insert into t2 values(1,2);
insert into t2 values(2,3);
insert into t2 values(3,4);
insert into t2 values(4,5);
insert into t2 values(5,1);
```

So the status of view0 is

```
t2_a,t2_b,t1_b
1,2,3
2,3,4
3,4,5
4,5,6
5,1,2
```

	Session1	**Session2**
1	Begin	
2		Begin
3	update t1 set b=12 where a=1;	
4		select * from view0 where t2_a>3 for update; +------+------+------+ | t2_a | t2_b | t1_b | +------+------+------+ | 5 | 1 | 2 | | 4 | 5 | 6 | +------+------+------+
5		Commit;(Success)
6	Commit;(Success)	

At third line TiDB locks the records of table t1 a = 1 until the sixth line releases the lock. Due to the nature of exclusive locks, the fourth line's attempt to acquire a lock on the record with a = 1 in table t1 is blocked, but TiDB grants the session2 lock.

The update statement locks data that does not exist

Data Found:

2020/06/12

Severity:

(S1)Critical

Test Case:

Drop database if exists db;

Create database db;

Use db;

Create table t(a int primary key, b int);

	Session1	**Session2**
1	Begin	
2		Begin
3	Update t set b=314 where a=1;(empty)	
4		Insert into t values(1,3);(blocking)
5	Commit;(success)	Insert into t values(1,3);(blocking)
6		Insert into t values(1,3);(success)
7		Commit;(Success)

The write operation of TiDB reads the latest submitted data, only locks the data that meets the conditions, but does not avoid the phantom (although the read operation can avoid the phantom through MVCC), then the write operation of the third line above will not lock the data, but in fact, TIDB locks it, blocking the insertion operation of another transaction

MySQL

Update BLOB data error

Data Found:

2020/05/02

Severity:

(S3)Non-Critical

Test Case:

Transaction ID	**Operation Start Timestamp**	**Operation End Timestamp**	**Operation Detail**	**State**
1	2020-08-05 15:52:27.477	2020-08-05 15:52:27.484	Update tablecsacas0 set attributeqwdcwq3=FILE(“./data_case/obj/12obj_file.obj”) where primarykeycqwda0 =	Success

			15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	
1	2020-08-05 15:52:27.495	2020-08-05 15:52:27.501	Update tablecsacas0 set attributeqwdcwq3=FILE(“./data_case/obj/12obj_file.obj”) and other column where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904	Success
1	2020-08-05 15:52:27.508	2020-08-05 15:52:27.512	Select attributeqwdcwq3 and other column from tablecsacas0 where primarykeycqwda0 = 15363173 and primarykeycqwda1 = 940396828 and primarykeycqwda2 = 1209414904 for update	Success and Return attributeqwdcwq3 = NULL (ERROE)

For BLOB data type, when the new value and the old value written by the update operation are for the same binary file, the value actually written is null and success is returned.

PostgreSQL

Write skew of SSI

Data Found:

2020/07/25

Severity:

(S1)Critical

Test Case:

Transaction ID	Operation Start Timestamp	Operation End Timestamp	Operation Detail	State
206	255567481387300	255567482259400	Select attribute1 from table_7_1 where primarykey=832	Success

204	255567479507200	255567480060500	Select attribute1 from table_7_4 where primarykey= 1460	Success
206	255567484738700	255567485188500	Update table_7_4 set attribute where primarykey=1460	Success
204	255567484625200	255567485012500	Update table_7_1 set attribute1 = -635092 where primarykey= 832	Success
204	255567485386900	255567485913000	Commit	Success
206	255567486411400	255567486923500	Commit	Success

Transaction 206 reads the record(832) of table_ 7_ 1， then transaction 204 writes a new record to cover it, so transactions 206 to 204 have a RW dependency. Similarly, transaction 204 reads the record(1460) of table_ 7_ 4, then transaction 206 writes a new record to cover it, so transactions 204 to 206 have a RW dependency. Finally, transactions 204 to 206 generate a circular dependency, that is, write skew. However, the test environment is the SSI isolation level of PostgreSQL, which is a bug.

Closed/Duplicate bug reports

MySQL

Data Found:

2020/09/25

Severity:

(S3)Non-Critical

MySQL Bug #69812