Resolving In-Doubt Transactions – by ilker taysi

Distributed transactions perform DML on multiple databases which is a bit more complicated task because the database must coordinate the consitency in those seperate or even perhaps between different DBMSs (like Oracle - MS SQL). To ensure the transaction atomicity, Oracle implements a 2-phase commit mechanism through which the distributed transactions undergo some phases like prepare, commit, forget, etc. This phases constitute the hand-shake mechanism of the distributed transaciton.

However, sometimes things may go wrong (due to some network, system problem or even a reconfiguration of the underlying objects) and one of the phases fails while others are ok. Here, we say that the transaction becomes in-doubt. Normallly this problem should be handled by the RECO process itself, but in some cases this cannot br performed.

Why RECO cannot perform in some cases?

One of the databases involved in the distributed transaction might be unreachable (netowrk, system issues etc.) while the RECO was trying to resolve the problem (even when retrying to recover). **(UNSTUCK)**

The lookup tables of the "2-phase commit" mechanism might become inconsistent with the transaction itself. **(STUCK)**

Handling UnStuck Transactions

Hopefully, there is no inconsistency between the lookup tables and the transaction and the following code resolves the problem:

To see the waiting transactions -> DBA_2PC_PENDING view.

```
SQL> select local_tran_id,global_tran_id, state,mixed, commit# from dba_2pc_pending;

97.33.166765    ORCL.781a8889.97.33.166765    prepared          no    60787107482
```

Here, **'97.33.166765'** is the transaction id of the distributed transaction, which will be used in the following commands.

If the state of the transaction is "prepared" and there is no inconsistency, the transaction can be forced to rollback, or maybe if the underlying problem which caused the in-doubt transaction is resolved the transaction can be forced to commit as follows:

```
SQL> ROLLBACK FORCE  '97.33.166765' /* ->replace with ur own trx_id */
or
SQL> COMMIT FORCE  '97.33.166765' /* ->replace with ur own trx_id */
```

Note: If the command hangs, go to the "Handling Stuck DBA_2PC_PENDING" section.

If the state of the transaction is "collecting" and you execute the above command, you may see an error like:

ERROR at line 1:
ORA-02058: no prepared transaction found with ID 97.33.166765

Execute the following command to purge the transaction

SQL> EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('97.33.166765'); /* ->replace with ur own trx_id */

PL/SQL procedure successfully completed.

Test to confirm that the transaction has gone

SQL> SELECT * FROM DBA_2PC_PENDING;

No rows returned.

The DBA_2PC_PENDING view is a lookup view, so it might be misleading sometimes. The actual transaction entry view is **X$KTUXE** ([**K**]ernel [**T**]ransaction [**U**]ndo Transa[**X**]tion [**E**]ntry).
This view gives info about the state of transactions that require transaction recovery:

X$KTUXE

COLUMN	TYPE	DESCRIPTION
ADDR	RAW	Address of this row in SGA
INDX	NUMBER	Index of this row in SGA
INST_ID	NUMBER	Instance Number
KTUXEUSN	NUMBER	Undo Segement #
KTUXESLT	NUMBER	Slot Number
KTUXESQN	NUMBER	Wrap Number
KTUXERDBF	NUMBER	Relative File
KTUXERDBB	NUMBER	Relative Block
KTUXESCNB	NUMBER	SCN Base prepare/commit
KTUXESCNW	NUMBER	SCN Wrap prepare/commit
KTUXESTA	VARCHAR2(16)	Transaction Status
KTUXECFL	VARCHAR2(24)	Transaction Flags
KTUXEUEL	NUMBER	Link to Coommit List

The concat of KTUXEUSN, KTUXESLT and KTUXESQN gives us the transacion number:

KTUXEUSN**.** KTUXESLT. KTUXESQN = **97.33.166765**

So, we can query the transaction view like:

```
SQL> SELECT * FROM X$KTUXE WHERE
        KTUXEUSN=97
AND KTUXESLT=33
AND KTUXESQN =166765;

No rows returned.
```

It should return no value, since the transaction has gone...

Handling Stuck Transactions

Our ultimate goal is not seeing the transaction in X$KTUXE table; and ensuring that the dictionary tables like PENDING_TRANS$ to be consistent with this information.

Stuck transactions can be examined under the below conditions:

Cond 1: ***DBA_2PC_PENDING view have entries about our transaction but there is no transaction in reality***

The condition is that; when we issue select to the dictionary views like the DBA_2PC_PENDING, PENDING_TRANS$, etc. we see the transaction, but the transaction does not exist in X$KTUXE view.

If the state of the transaction (in DBA_2PC_PENDING) is committed, rollback forced or commit forced then it can be cleaned by:

```
SQL> EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('97.33.166765'); /* ->replace with ur own trx_id */

PL/SQL procedure successfully completed.
```

If the state of the transaction is prepared, we have to clean manually as follows:

```
SQL> DELETE FROM SYS.PENDING_TRANS$ WHERE LOCAL_TRAN_ID = '97.33.166765'; /* ->replace with ur own trx_id */
SQL> DELETE FROM SYS.PENDING_SESSIONS$ WHERE LOCAL_TRAN_ID ='97.33.166765' ; /* ->replace with ur own trx_id */
SQL> DELETE FROM SYS.PENDING_SUB_SESSIONS$ WHERE LOCAL_TRAN_ID = '97.33.166765'; /* ->replace with ur own trx_id */
SQL> COMMIT;
```

Note: DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY command does not purge the actual transaction in X$KTUXE view. It touches the dictionary views.
To really purge the transaction, we need a commit or rollback. In the above case, however, there is no real transaction so we don't need to worry about rollback or commit, all we do is cleaning the dictionary...

Cond 2: ***DBA_2PC_PENDING view does NOT have entries about our transaction but there IS A transaction***

This is something like a orphan transaction that the dictionary is not aware of.

Trying to force commit or rollback this transaction may result in error like below, since the dictionary is not aware:

```
SQL> ROLLBACK FORCE  '97.33.166765' /* ->replace with ur own trx_id */

ORA-02058: no prepared transaction found with ID 97.33.166765
```

What we need to do at this point is; recovering our transaction from being an orphan by inserting some dummy records into dictionay tables (so the views...) and then force a rollback or commit:
You do not have to change the parameters in the insert command other than the transaction id.

```
SQL> ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;

SQL> INSERT INTO PENDING_TRANS$ (LOCAL_TRAN_ID, GLOBAL_TRAN_FMT, GLOBAL_ORACLE_ID,
STATE, STATUS, SESSION_VECTOR, RECO_VECTOR, TYPE#, FAIL_TIME,RECO_TIME)
VALUES
(
   '97.33.166765',   /* ->replace with ur own trx_id */
   306206,
   'XXXXXXX.12345.1.2.3',
   'prepared','P',
   hextoraw( '00000001' ),
   hextoraw( '00000000' ),
   0, sysdate, sysdate
);


SQL> INSERT INTO PENDING_SESSIONS$
VALUES
(
    '97.33.166765',  /* ->replace with ur own trx_id */
    1, hextoraw('05004F003A1500000104'),
    'C', 0, 30258592, '',
    146
);


COMMIT;
```

Now, we should be able to rollback or commit.

```
SQL> ROLLBACK FORCE  '97.33.166765' /* ->replace with ur own trx_id */
```
or
```
SQL> COMMIT FORCE  '97.33.166765' /* ->replace with ur own trx_id */
```

Lastly, we remove the dummy entry from the dictionary:

```
SQL> ALTER SYSTEM ENABLE  DISTRIBUTED RECOVERY;

SQL> ALTER SYSTEM SET "_smu_debug_mode" = 4;

SQL> COMMIT;

SQL>  EXECUTE  DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('97.33.166765');  /* ->replace with
ur own trx_id */

SQL> ALTER SYSTEM SET "_smu_debug_mode" = 0;

SQL> COMMIT;
```

Check to see whether the transaction has gone:

```
SQL> SELECT * FROM X$KTUXE WHERE
        KTUXEUSN=97
AND KTUXESLT=33
AND KTUXESQN =166765;

No rows returned.
```

Cond 3: ***DBA_2PC_PENDING has entry and there is a transaction but COMMIT or ROLLBACK HANGS!***

In the situation, where COMMIT FORCE or ROLLBACK FORCE hangs,

Trying to purge the transaction will give an error like:

```
SQL> EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('97.33.166765'); /* ->replace with ur own trx_id */

ERROR at line 1:
ORA-06510: PL/SQL: unhandled user-defined exception
ORA-06512: at "SYS.DBMS_TRANSACTION", line 94
ORA-06512: at line 1
```

Solution: The solution is the combination of Cond1 and Cond2:

First, delete the dictionary entries.

```
SQL> DELETE FROM SYS.PENDING_TRANS$ WHERE LOCAL_TRAN_ID = '97.33.166765'; /* ->replace with ur own trx_id */
SQL> DELETE FROM SYS.PENDING_SESSIONS$ WHERE LOCAL_TRAN_ID ='97.33.166765' ; /* ->replace with ur own trx_id */
SQL> DELETE FROM SYS.PENDING_SUB_SESSIONS$ WHERE LOCAL_TRAN_ID = '97.33.166765'; /* ->replace with ur own trx_id */
SQL> COMMIT;
```

Then, insert dummy record, force commit and finally purge the transaction.

```
SQL> ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;

SQL> INSERT INTO PENDING_TRANS$ (LOCAL_TRAN_ID, GLOBAL_TRAN_FMT, GLOBAL_ORACLE_ID,
STATE, STATUS, SESSION_VECTOR, RECO_VECTOR, TYPE#, FAIL_TIME,RECO_TIME)
VALUES
(
   '97.33.166765',   /* ->replace with ur own trx_id */
   306206,
   'XXXXXXX.12345.1.2.3',
   'prepared','P',
   hextoraw( '00000001' ),
   hextoraw( '00000000' ),
   0, sysdate, sysdate
```

```
);


SQL> INSERT INTO PENDING_SESSIONS$
VALUES
(
    '97.33.166765',  /* ->replace with ur own trx_id */
    1, hextoraw('05004F003A1500000104'),
    'C', 0, 30258592, '',
    146
);


COMMIT;

SQL> COMMIT FORCE  '97.33.166765' /* ->replace with ur own trx_id */

SQL>  EXECUTE  DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('97.33.166765');  /*  ->replace  with
ur own trx_id */
```