# Persistent Transaction Context

# proposal

Change History:

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Name | Description | Distributed to |
| 5/19/2014 | Ping Zhou | Initial draft | Carl, Nick,Derek |
| 5/19/2014 | Derek Reiger | Review + comments | Ping, Nick, Carl |
| 5/20/2014 | Ping Zhou | Response to Derek’s comments | Carl, Nick,Derek |
|  |  |  |  |

Table of Contents

[Persistent Transaction Context 1](#_Toc388974858)

[proposal 1](#_Toc388974859)

[1. Background 2](#_Toc388974860)

[2. Design 2](#_Toc388974861)

[3. Statistics 5](#_Toc388974862)

[4. Locator format change 6](#_Toc388974863)

[5. Exceptions 6](#_Toc388974864)

[6. Persist-able transaction context 6](#_Toc388974865)

[7. Persisted fields 7](#_Toc388974866)

[8. Repository changes 9](#_Toc388974867)

[9. Truncation 9](#_Toc388974868)

[10. Document 10](#_Toc388974869)

[11. Long-term 11](#_Toc388974870)

[12. Test 11](#_Toc388974871)

1. Background

  P66 has another issue we would like to address as quickly as possible – their SAP system has long-running transactions (> 3 days) that prevent the truncation point from moving, and that is unacceptable for that amount of time given the high volume and amount of log generated.  They don’t want to keep that much archive log around in case RAO needs to restart, and while they are investigating whether the long-running transactions are expected, they are asking for functionality out of RAO to resolve this.

To address it, Derek has been thinking about transaction context persistence solution and here is a high-level overview:

* In addition to the open transaction list in Operation Processor, keep a persisted open transaction list that is written to the RASD on some configurable interval
* On persistence, iterate through the open transaction list either to the end of the list or up to the first “non-persist-able” transaction context
  + The persistence of the transaction context will likely need to be determined since, for example, the transaction context could be in a state where there are pending operations, and I’m thinking we may need to just have some cases where the transaction context cannot be persisted
* Write all persist-able transaction contexts to the RASD
* Update the locator truncation point to the greater of either the SCN of the begin operation for oldest open transaction (OATSCN) or the SCN of the begin operation for the last transaction that was persisted (LPATSCN)
* During operation processing, keep the LPATSCN/OATSCN in-synch with operation processing, adjusting the truncation point as necessary (e.g. if the commit for the transaction referenced by the LPATSCN is processed, use the current OATSCN until the next persistence.
* On replication re-start, the open transaction lists are restored based on the last persisted state

With this solution, the truncation point can move (as long as the transaction contexts are “persist-able”) even when there are long-running transactions in the PDB.

.

1. Design

Current processed L3

LastPersisted L1

RS LTM locator L0

T3

T2

T1

Last Active T4

RS LTM locator L2

OAT

LatestActive T5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OP | SCN | OATSCN | LPATSCN | Trunc Pt SCN |
| OAT | 1 | 1 | Null | 1 |
| T1 Begin | 2 | 1 | Null | 1 |
| T2 Begin | 3 | 1 | Null | 1 |
| T3 Begin | 4 | 1 | Null | 1 |
| T4 Begin | 5 | 1 | Null | 1 |
| T2 Commit | 6 | 1 | Null | 1 |
| PERSIST L1 | T3 is non-persistable { OAT(SCN=1), T1(SCN=2) }, OATSCN=1, LPATSCN=2 | | | |
| T4 Commit | 7 | 1 | 2 | 2 |
| T5 Begin | 8 | 1 | 2 | 2 |
| T1 Commit | 9 | 1 | 2 | 2 |
| T3 Commit | 10 | 3 | 2 | 3 |
| T5 Commit | 11 | 7 | 2 | 7 |

OAT = oldest active transaction = long running transaction

T1= Last persist-able-active-transaction // the persistence process should be stopped/finished when non-persist-able transaction occur, so here T1 is last persist-able active transaction.

LAPSCN = Last persist-able-active-transaction begin SCN = Next start scan point = Truncation point

L0 = Last RS LTM locator when persistent transaction context occur.

L1 = Last persistent transaction context locator.

L2 = Latest RS LTM locator.

L3 = Current processed record locator.

T5 = Latest persist-able-active-transaction

Last active transaction list = {OAT, T1, T3, T4}

Last persist-able active transaction list = {OAT, T1} // assume T3 is non-persist-able then T4 is also taken as non-persist-able

Latest active transaction list = {OAT, T5}

We cannot start from latest active transaction begin or latest RS LTM locator, because we only keep last active transactions list, we cannot guarantee that no new open transaction occur after last active transaction list persisted.

For short-term, I agree that we can just simply start from Last active persist-able transaction begin SCN (LAPSCN).

Update the locator truncation point to the greater of either the SCN of the begin operation for oldest open transaction (OATSCN) or the SCN of the begin operation for the last transaction that was persisted (LPATSCN)

During operation processing, keep the LPATSCN/OATSCN in-synch with operation processing, adjusting the truncation point (originally OAT field) as necessary (e.g. if the commit for the transaction referenced by the LPATSCN is processed, use the current OATSCN until the next persistence

RAO will always restore the open transaction lists based on the last persisted state and re-start from the truncation point SCN. (The truncation point SCN will be always increase, never decrease).

The reason for why we had to lock the operation processor and we do not persist according to time interval is we want to guarantee the same locator (same TRUNC\_PT\_SCN) even on re-scan, then we can avoid any locator format issue.

For example,

If we persist according to the time interval then:

 T0          T1           T2         P1           T3          P2       T4        T5(non-persist)        RS Loc          P3

|              |               |            |               |           |          |                      |           |                     |

-----------------------------------------------------------------------------------------------------------------------------🡪

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Persist-able-active transactions | LAPSCN | Locator | RS locator |
| 1 | T0,T1, T2 | T2begin=2 | 121112 | 121112 |
| 2 | T0,T1 | T2begin=2 | 131122 | 131122 |
| 3 | T0,T1,T2, T3, T4 | T4begin=4 | 151134 , 151154,151174 | 151154 |

Note: P1,P2,P3 the persist point. The truncation point should only increase, never decrease.

Next time during rescan, Start from T4, the time interval and the transaction count had been reset to zero, then the next persistence point maybe different, so T5 may be involved as persist-able:

 T0          T1           T2         P1           T3          P2       T4        T5(persist-able)        RS Loc          P3

|              |               |            |               |           |          |                      |           |                     |

-----------------------------------------------------------------------------------------------------------------------------🡪

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Persist-able active transactions | LAPSCN | Locator | RS Loc | Comments |
| 3 | T1,T2, T3,T4,T5 | T5begin=5 | 151134, 151145,151155,151175 | 151154 | 151155 (original 151154) should be filtered, but not |

During rescan the time interval and the transaction count had been reset to zero,When we still use the it to determine whether to persist transaction, the persistence point maybe different with the original one.

In order to guarantee the persistence point consistence even on rescan, we need keep the original transaction count for last persisted open transaction then the transaction count keep consistence even on rescan.

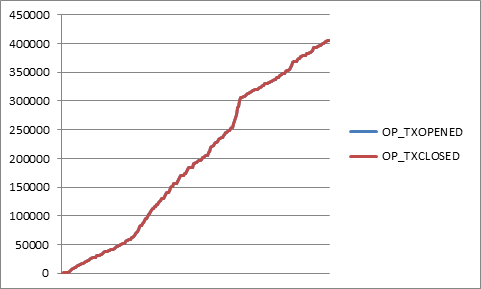
If we do not lock the operation processor:

Then even the persistence progress kick off at same place like every 1000 transactions when rescan, we cannot guarantee the transaction context (persist-able?, rollback stack? Etc) is same when this transaction context is really persisted.

Say, we have T1, T2,T3,T4 need to be persisted one by one, but when you persist T2,  the T3, T4 may or may not had changed for each rescan.

As a result, Active transaction context list will be persisted in-sync with operations process according to interval of every X transaction processed, for example, TransCtxPersistenceInterval = 100000 transactions. Which means during the persistence the operation processor will be locked.

  Here is the last STATRACK sample Derek got from P66 for the number of transactions processed (this included the LM where clause filter):



This was about 4 hours’ worth of data, so if we set the interval to persist every 100,000 transactions by default, then persistence is only going to occur once an hour, and I don’t expect that to impact operation processing performance.  We will of course need some statistics to monitor it, and some commands to provide information on it.

The reason for why not use time interval instead of transaction count to identify the persistent point?

The timestamp field from v$logmnr\_contents like ‘2014-12-01 09:46:58.0’ is not accurate enough to identify a single transaction, several transactions may occur in just same time,

TIMESTAMP SQLREDO

------------------------------ ---------------------------------

2014-12-01 09:47:04.0 set transaction read write;

2014-12-01 09:47:04.0 update "SYS"."JOB$" …

2014-12-01 09:47:04.0 commit;

2014-12-01 09:47:04.0 set transaction read write;

2014-12-01 09:47:04.0 select \* from "SYS"."DBMS\_LOCK\_ALLOCATED" …

2014-12-01 09:47:04.0 update "SYS"."DBMS\_LOCK\_ALLOCATED" set …

2014-12-01 09:47:04.0 commit;

1. Statistics
2. Total transaction context persistence count
3. Total transaction context persistence time
4. Last transaction context persistence time
5. Locator format change

After turn on the transaction persistence feature, actually RS will not return the locator with an OATSCN field any longer … RS will return the truncation point locator, and with this change, the OATSCN field is now the TRUNC\_PT\_SCN field. We will need to re-format the locator with this functionality to change the OATSCN to TRUNC\_PT\_SCN and make it least significant, even less than LOC\_ID

GENID + SCN + SCNGENID + THREAD + LSN + BLK + OFF + LOCID + TRUNC\_PT\_SCN

Actually if we can guarantee the TRUNC\_PT\_SCN keep same even on re-scan for same record, then we do not need to take care of the sequence for LOCID and TRUNC\_PT\_SCN, so for short-term, we should first try to avoid change the sequence of OATSCN and just change the name from OATSCN to TRUNC\_PT\_SCN in order to keep migration simple.

GENID + SCN + SCNGENID + THREAD + LSN + BLK + OFF + TRUNC\_PT\_SCN + LOCID

1. Exceptions
2. Persist-able transaction context

Persist-able when:

1. All DDL transactions is non-persist-able.

Note: because DML transaction will be hold until DDL transaction processed, so we can just ignore DDL transaction and process next DML transaction.

1. All pending insert/delete (\_pendingInsertRecords/\_pendingDeleteRecords) had been consumed.
2. All unprocessed records processed.
3. All pending duplicated begin operations (\_pendingDuplicatedBeginRecord) had been consumed.
4. All active chained operations had been consolidated to one operation (\_activeChainedOp).
5. DML transactions do not contain the operations on tables with LOB column, even those LOB columns are not enabled to be replicated, LOB status(like LOB end etc) still need to be maintained. Even no LOB column update, \_priorLOBRecord still need to be maintained.

Note: LOB context persistent is a big hole, we need care about the LOB process status and even LOB data, for simplicity, I suggest to skip all LOB related for short-term.).

1. DML transactions do not in the progress of “rollback to savepoint”(pendingRollback is not null).
2. DML transactions which rollback context size does not exceed a threshold for example, \_rollbackRcdsThreshold = 10000?
3. No too many open transactions for example, exceed maxPersistTransCtx = 100, persistence should be terminated.
4. All active procedures had been processed.
5. The SCN gap between the persist-able active transaction begin and the current processed transaction begin exceeds a threshold to judge whether it is a long-running transaction.

T4

P1

T3

T2

T1

OAT

Can we also count the SCN gap between the persist-able active transaction begin and the current processed record

(transCtxPersistenceSCNThreshold, trans\_ctx\_persist\_scn\_threshold), if the gap is too small, say transCtxPersistSCNThreshold =100 then we stop to persist this transaction context,

For example,

Even T3, T4 is persist-able, but because the SCN gap between begin SCN and the current processed record is too small, only T1 will be persisted.

1. Persisted fields
2. Transaction Context

|  |  |  |
| --- | --- | --- |
| Field | Type | Comments |
| \_begin | LogMinerRecord | Persist and used to re-build the transactionContext, \_isMaintTran,\_isRecursive, sessionNotFound,\_session, \_tranId,\_thread |
| \_transactionProcessor | DMLTransactionProcessor | We can just persist required fields like RollbackContext and rebuild DMLTransactionProcessor. |
| \_beginSentOut | Boolean | Persist |
| \_objectMap? |  | Not persist |
| \_parentTransaction |  | Rebuild during session initialized |
| \_operationProcessor, \_standbyScanner |  | Not Persist, rebuild. |
| \_isDDLTrans |  | Because a DDL transaction is  non-persist-able |
| \_pendingInsertRecords/  \_pendingDeleteRecords |  | Because a transaction with any pending records is non-persist-able |
|  |  |  |

1. LogMinerRecord

|  |  |  |
| --- | --- | --- |
| Field | Type | Comments |
| \_row | LogMinerRow | Persist; Rebuild the whole LogMinerRecord, only for Begin operation now. |

1. LogMinerRow

|  |  |  |
| --- | --- | --- |
| Field | Type | Comments |
|  |  | For simplicity, persist all the fields, currently it only target for Begin Record. |

1. DMLTransactionProcessor

|  |  |  |
| --- | --- | --- |
| Field | Type | Comments |
| \_rollbackCtx | **RollbackContext** | **Persist,** **almost all the information for RollbackContext should be persisted, for simplicity, just persist all RollbackContext** |
| \_replicableOperation | **Long** | **Persist** |
| \_logRecHasMarkedObjects |  | **Not Persist, rebuild** |
| \_operationsProcessed, \_operationsSkipped |  | **Just indicate the statistics for current round process, not persist.** |
| \_ltm \_session  \_isOracleVersionGreaterThan12cR1 |  | Not persist, Rebuild. |
| \_lobHelpCtx |  | **Not persist, Rebuild** |
| \_activeChainedOp |  | **Rebuild to null** |
| \_commit |  | **DDL only, not persist** |
| \_isOracleVersionLessThan11gR2  \_isOracleVersionLessThan11gR1  \_isOracleVersionGreaterThan12cR1  \_conninfo  \_filterThreadIDsList  \_transCtx |  | **Rebuild, Not persist** |
|  |  |  |

1. **RollbackContext**

|  |  |  |
| --- | --- | --- |
| Field | Type | Comments |
| \_rollbackMetaStack | **List< RollbackStackElement>** | **Persist** |
| top | **RollbackStackElement** | **Persist** |
| \_currentRID | **int** | **Persist** |
| \_firstRID | **int** | **Persist** |
| \_pendingRollback |  | **\_pendingRollback is very common case for rollback to savepoint, but if for short-term, I also agree that we can ignore the transaction in the middle of rollback to save point.** |
| \_pendingRollbackLocator |  | **Not Persist** |
| \_transProcessor |  | **Rebuild** |

1. Repository changes
2. Store all persist-able-active transaction contexts list as a whole item.

For example,

Add new name,param pair in the already existed raSysParam RASD table.

But the down side is we need to persist the persist-able-active transaction context even it had been persisted through last persistence round.

1. Add new RASD table like :

Create table ra\_transCtxList

{

xid bigint primary key,

beginRow long varbinary,

\_beginSentOut int,

\_replicableOpCnt int,

\_transCount bigint,

\_oatSCN bigint,

rollbackcontext long varbinary

}

Then if any new transaction context need to be persisted, we just need insert item, and we can easy to truncate/delete the already commit one.

We all prefer to #2

So far, the speed is acceptable for big transaction.

|  |  |
| --- | --- |
| Rollback stack size | Time spent |
| 10000 | 1.6s |
| 50000 | 4.6s |
| 100000 | 9.8s |

And I also test many small transactions with only one record each.

|  |  |
| --- | --- |
| transactions | Time spent |
| 1000 | 1.1s |
| 10000 | 4.9s ~ 5.6s |
| 20000 | 9.6s |
| 50000 | Locked? ASA has limitation for insertion? |

I also use the Externalizable instead of Serializable to speed up the persist of rollbackcontext,

1. The size comparison

|  |  |  |
| --- | --- | --- |
|  | Serializable | Externalizable |
| One rollback with 100000 items | 13869819 | 10678020 |
| 10000 rollback with 1 item each | 6720000 | 3230000 |

1. The speed comparison

|  |  |  |
| --- | --- | --- |
|  | Serializable | Externalizable |
| One rollback with 100000 items | ~9s | 3.5s |
| 10000 rollback with 1 item each | 5.2s | 4.9s |

For large transaction, Externalizable will speed up the persistence, but for small transaction, almost no change, but it can save the disk space.

1. Truncation
2. The persisted active transaction context list should also be truncated as part of RASD truncation process (also part of persistence process? Next step?) according to the oat SCN for persisted last active persist-able Transaction.

LAPT=truncation point

OAT

Persisted LAPT

(Keep OAT)

The OATSCN will be keep into each persisted last active persist-able transaction context, when truncation occur,

1. If the persisted last active persist-able transaction context can be loaded according to the truncation point, all the persisted last active persist-able transaction contexts which version (begin locator) smaller than this OATSCN fetched from persisted last active persist-able transaction context will be removed.

Truncation point

Persisted LAPT

(Keep Old OAT)

LAPT

Old OAT

1. If truncation point even larger than the last transaction context stored in oracletransactioncontext table, all the persisted last active persist-able transaction contexts which version (begin locator) smaller than this truncation point will be removed(actually all will be removed).

In order to make the truncation easier, we plan to persist all open persist-able transaction when persistence threshold matched and we will take the persistence point locator as the persistence version for all those open persist-able transactions [1], even this transaction keep same as old one. The downside is duplicated. But we can easily truncate the entire old persisted transaction context according to version.

1. Rollback stack issue

For example,

Rollbackcontext = {1,2,3,4,5,6}

Top = 6

LAPSCN = 4

RA will rescan from 4 and rebuild the entries. Then the operation with locator {4,5,6} should skip to push to rollback stack to avoid duplicate, That is to say, only the operation with locator larger than the current top element’s locator can be pushed to stack.

1. Related commands

pdb\_truncate\_xlog

* Truncate the database transaction log according to truncation point

rasd\_trunc\_schedule force

* Truncate repository according to the truncation point, but for oracletransactioncontext table should be truncated according the OATSCN for last active persist-able transaction context.
* ra\_truncate\_tctx

1. Document
2. New parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Default | Range | Description |
| lr\_persist\_tctx | True | True  False | Determines whether open transaction persistence is used for long running transactions. |
| lr\_persist\_lrt\_threshold | 3600s | 1~Long.MAX | When open transaction persistence is enabled, determines the length of time a transaction must be open for to be considered long-running. |
| lr\_persist\_records\_threshold | 10000 | 1~Long.MAX | Indicates the maximum numbers of record contained in a persist-able active transaction context. |
| lr\_persist\_tctx\_threshold | 100000 | 1~Long.MAX | When open transaction persistence is enabled, determines the number of transactions that trigger an open transaction persistence event. |

1. New statistics

LR Total transaction context persistence count

1

LR Total transaction context persistence time (ms)

35479

LR The last transaction context persistence time (ms)

6735

LR Time open transactions persistence last started

Fri May 30 11:36:31 CST 2014

1. Changed command.

The output for ra\_helplocator on Logminer-based RAO had been changed.

1> ra\_helplocator 0000000003f369630000000100001379000000000000000003f20c3900000000

2> go

Field Hex Value Decimal Value

------------- --------------- -------------

GENID 0x0000 0

SCN 0x000003f36963 66283875

SCNGENID 0x0000 0

THREAD 0x0001 1

LSN 0x00001379 4985

BLKNUM 0x00000000 0

BLKOFFSET 0x0000 0

TRUNC\_PT\_SCN 0x000003f20c39 66194489

LOCID 0x00000000 0

Note: The OATSCN had been changed to TRUNC\_PT\_SCN, because the truncation point can move, even the OAT(oldest active transaction)SCN stick(not change).

1. New commands.

* List the entire persisted transaction contexts in RASD briefly

1> ra\_help\_tctx

2> go

XID Begin sent out Replicable operations

Version

------------------- -------------- ---------------------

-----------------------------------------------------------------

0000.0001.00000001 FALSE 0

0000000003f369630000000100001379000000000000000003f20c3900000000

XID: truncation id

Begin sent out: whether such transaction had been sent out.

Replicable operations: how many replicable operations contains in such transaction context.

Version: The locator of begin operation for the latest transaction when the transaction context persistence occur, which used to identify a bunch of long-running transactions context persisted in such time.

* Truncate the persisted transaction context from RASD.

1> ra\_truncate\_tctx 0000000003f369630000000100001379000000000000000003f20c3900000000

2> go

Msg 0, Level 20, State 0:

Server 'raolgsp120', Procedure 'ra\_truncate\_tctx 0000000003f369630000000100001379000000000000000003f20c3900000000', Line 1:

successful

(0 rows affected)

1>

Here 0000000003f369630000000100001379000000000000000003f20c3900000000 is the truncation point come from RepServer.

1. Long-term
2. I recommend providing user interface to manually persist active transaction context list. User can force to do it when no valid persisted transaction context list or user wants to do truncation right now.
3. We’d better keep the last RS LTM Locator after transaction context persisted, then next time, RA can start from this persisted last RS LTM Locator, we even can truncate the active list according to this last RS LTM Locator when we fetched from RS after persisting the transaction context.
4. If the transaction context persisted spend too much time, for example, even affect next round persistent, the last persistent will be stopped immediately.
5. Test

select \* from qaxuser.tab1

create table tab1 (a int, b char(10));

truncate table qaxuser.tab1

1. Open transaction but not commit then issue ra\_marker

insert into qaxuser.tab1 values(5,'rollback');

On RA issue:

1> ra\_marker "admin version"

2> go

0000000003f5e8a0000100010000138700000d1e0010000003f5e8a000000000 \_bg \_tr for ~"(QAXUSER osid 200

0000000003f5e8a0000100010000138700000d1e0010000003f5e8a000000001, \_ap owner=~"(QAXUSER ~"%TAB1.rs\_insert

0000000003f5e8ad000000010000138700000d350170000003f5e8ad00000000, \_bg \_tr for ~",RAOUSER\_DBA osid 133

0000000003f5e8ad000000010000138700000d350170000003f5e8ad00000001, admin version

0000000003f5e8b0000000010000138700000d3b0110000003f5e8b000000000, \_cm \_tr

0000000003f5f1fc00000001000013870000215d01d0000003f5f1fc00000000, \_bg \_tr for ~",RAOUSER\_DBA osid 133

0000000003f5f1fc00000001000013870000215d01d0000003f5f1fc00000001, admin version

0000000003f5f1fe000000010000138700002161018c000003f5f1fe00000000, \_cm \_tr

1. Generate same locator when rescan

connect source lti raolgsp120.po112w 740

get maintenance user for raolgsp120.po112w

get truncation raolgsp120.po112w

0000000003f6122a00000001000013870000596f0180000003f6122400000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f61224000a000200013fe9 \_bg \_tr for ~",RAOUSER\_DBA osid 72

0000000003f6122a00000001000013870000596f0180000003f6122400000001,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f61224000a000200013fe9 admin version

0000000003f6122c0000000100001387000059730118000003f6122400000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f61224000a000200013fe9 \_cm \_tr

0000000003f622bd0000000100001387000080160118000003f622b600000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f622b6000200150001419b \_bg \_tr for ~",RAOUSER\_DBA osid 69

0000000003f622bd0000000100001387000080160118000003f622b600000001,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f622b6000200150001419b admin version

0000000003f622c000000001000013870000801b00d0000003f622b600000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f622b6000200150001419b \_cm \_tr

0000000003f624cc0000000100001387000082b40044000003f624c500000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f624c50001000500013ec8 \_bg \_tr for ~",RAOUSER\_DBA osid 69

0000000003f624cc0000000100001387000082b40044000003f624c500000001,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f624c50001000500013ec8 admin version

0000000003f624ce0000000100001387000082b80010000003f624c500000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f624c50001000500013ec8 \_cm \_tr

When rescan:

1> ra\_locator

2> go

Locator

-----------------------------------------------------------------

0000000003f6122c0000000100001387000059730118000003f6122400000000

connect source lti raolgsp120.po112w 740

get maintenance user for raolgsp120.po112w

get truncation raolgsp120.po112w

0000000003f6122a00000001000013870000596f0180000003f6122400000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f61224000a000200013fe9 \_bg \_tr for ~",RAOUSER\_DBA osid 72

0000000003f6122a00000001000013870000596f0180000003f6122400000001,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f61224000a000200013fe9 admin version

0000000003f6122c0000000100001387000059730118000003f6122400000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f61224000a000200013fe9 \_cm \_tr

0000000003f622bd0000000100001387000080160118000003f622b600000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f622b6000200150001419b \_bg \_tr for ~",RAOUSER\_DBA osid 69

0000000003f622bd0000000100001387000080160118000003f622b600000001,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f622b6000200150001419b admin version

0000000003f622c000000001000013870000801b00d0000003f622b600000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f622b6000200150001419b \_cm \_tr

0000000003f624cc0000000100001387000082b40044000003f624c500000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f624c50001000500013ec8 \_bg \_tr for ~",RAOUSER\_DBA osid 69

0000000003f624cc0000000100001387000082b40044000003f624c500000001,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f624c50001000500013ec8 admin version

0000000003f624ce0000000100001387000082b80010000003f624c500000000,6 ~,B{0x}72616f6c6773703132302e706f313132770000000003f624c50001000500013ec8 \_cm \_tr

1. Large transaction persistence
2. Truncate transaction persistence

1> ra\_help\_tctx

2> go

XID Begin sent out Replicable operations

Version

----------------- -------------- ---------------------

-----------------------------------------------------------------

281612415745928 TRUE 2

0000000003f68e55000100010000138a0000257901cc000003f68c4b00000000

281616710713230 FALSE 0

0000000003f68684000000010000138a00001c500010000003f6868400000000

563078802522533 FALSE 0

0000000003f685b8000000010000138a00001b6e0010000003f685b800000000

844467879886896 FALSE 0

0000000003f6859e000000010000138a00001b4d0010000003f6859e00000000

1125934266662728 FALSE 0

0000000003f68a2c000000010000138a000020870010000003f688ca00000000

1407396358472234 FALSE 0

0000000003f68e55000100010000138a0000257901cc000003f68c4b00000000

1688970119430255 FALSE 0

0000000003f68c4b000000010000138a000023770010000003f68a2c00000000

2533330625052818 FALSE 0

0000000003f68440000000010000138a000018df0010000003f6844000000000

2533395049562257 FALSE 0

0000000003f688ae000000010000138a00001f070010000003f688ae00000000

2814771242024976 FALSE 0

0000000003f686e8000000010000138a00001c880010000003f686e800000000

(10 rows affected)

1> ra\_truncate\_tctx 0000000003f68965000000010000138a00001f86018c000003f688ca0000

0000

2> go

Msg 0, Level 20, State 0:

Server 'raolgsp120', Procedure 'ra\_truncate\_tctx 0000000003f68965000000010000138

a00001f86018c000003f688ca00000000', Line 1:

successful

(0 rows affected)

1> ra\_help\_tctx

2> go

XID Begin sent out Replicable operations

Version

----------------- -------------- ---------------------

-----------------------------------------------------------------

281612415745928 TRUE 2

0000000003f68e55000100010000138a0000257901cc000003f68c4b00000000

1125934266662728 FALSE 0

0000000003f68a2c000000010000138a000020870010000003f688ca00000000

1407396358472234 FALSE 0

0000000003f68e55000100010000138a0000257901cc000003f68c4b00000000

1688970119430255 FALSE 0

0000000003f68c4b000000010000138a000023770010000003f68a2c00000000

(4 rows affected)

1>

1. Rollback after restart from truncation point

insert into qaxuser.tab1 values(5,'rollback');

savepoint a;

insert into qaxuser.tab1 values(6,'rollback');

insert into qaxuser.tab1 values(7,'rollback');

insert into qaxuser.tab1 values(8,'rollback');

0000000003f9d86300010001000013a200000e0f0010000003f9d86300000000, \_bg \_tr

0000000003f9d86300010001000013a200000e0f0010000003f9d86300000001, rs\_insert 5

0000000003f9d86500010001000013a200000e130060000003f9d86300000000, rs\_insert 6

0000000003f9d86700010001000013a200000e1601a8000003f9d86300000000, rs\_insert 7

0000000003f9d86800010001000013a200000e1a00e4000003f9d86300000000, rs\_insert 8

update the LTM locator

1> ra\_locator update

2> go

Msg 0, Level 20, State 0:

Server 'raolgsp120', Procedure 'ra\_locator update', Line 1:

successful

(0 rows affected)

1> ra\_locator

2> go

Locator

-----------------------------------------------------------------

0000000003f9da8100000001000013a2000019410168000003f9d99a00000000

Restart the RA from truncation point:

Issue insert again and rollback to save point:

insert into qaxuser.tab1 values(9,'rollback');

rollback to a;

0000000003f9dd5a00000001000013a200001d650010000003f9dd5a00000000, rs\_insert 9

0000000003f9dd7900000001000013a200001d890010000003f9dd7900000000, \_rl from 0x0000000003f9d86500010001000013a200000e130060000003f9d86300000000 to 0x0000000003f9dd7900000001000013a200001d890010000003f9dd7900000000