The Gains and Pains of Nologging Operations (Doc ID 290161.1)

** checked for relevance '23-Nov-2015' **

Checked for relevance on 22-May-2014

THE GAINS AND PAINS OF NOLOGGING OPERATIONS

Overview

Whereas a logged INSERT operation has to generate redo for every change data or undo block, nologging operations indicate that the database operation is not logged in the online redo log file. Even though a small invalidation redo record[1] is still written to the online redo log file, nologging operations skip the redo generation of the corresponding inserted data. Nologging can be extremely beneficial for the following reasons:

- data written to the redo is minimized dramatically
- time to insert into a large table or index or LOB can be reduced dramatically
- performance improves for parallel creation of large tables or indices

However, NOLOGGING is intended for configurations in which media recovery or the recovery of the corresponding object is not important. Thus, if the disk or tape or storage media fails, you will not be able to recover your changes from the redo because the changes were never logged.

Nologging operations are invoked by any of the following:

- SQL*Loader direct load operations
- Direct load INSERT operations from CREATE TABLE | INDEX or INSERT commands
- Loading into an object containing LOB data when its object's segment characteristic is NOCACHE NOLOGGING

For databases in ARCHIVELOG mode, nologging operations can only occur for a particular object if and only if:

- Database allows for nologging (ALTER DATABASE NO FORCE LOGGING) and
- Tablespace allows for nologging (ALTER TABLESPACE <NAME> NO FORCE LOGGING) and
- Object allows for nologging (ALTER TABLE <NAME> NOLOGGING)

This paper will cover the following topics:

- examples of nologging operations
- prevention of nologging operations
- detection of nologging operations on the primary and standby databases
- repair of nologged changes on the physical and logical standby databases

Examples of nologging operations

Below is a list of examples that can be used for testing purposes. The database must be in ARCHIVELOG mode and the segment must explicitly be set to NOLOGGING and must allow nologging operations to see the effect of nologging changes:

1. `insert /*+ APPEND */ into scott.emp select * from sys.emp2;`
2. `create table emp nologging as select * from sys.emp;`
3. `create index emp_i on emp(empno) nologging;`
4. sqlload operation with unrecoverable option

Prevention of nologging operations

When a standby database exists or if you want all transactions to be recoverable on a database, tablespace or object-wide perspective, it is recommended that you prevent nologging operations by issuing the relevant options. These options include:

- `ALTER DATABASE FORCE LOGGING` (database level) or
- `ALTER TABLESPACE <NAME> FORCE LOGGING` (tablespace level) on the relevant tablespaces you want to protect or

- `[CREATE | ALTER] TABLE <NAME> LOGGING` (example of object level) on the relevant objects you want to protect

This ensures that all transactions are logged and can be recovered through media recovery or Redo Apply or SQL Apply assuming appropriate data type support.

Force Logging is not required to use Data Guard. However without it, DBAs should expect additional administration in maintaining the affected objects. The next sections will cover detection and correction.

Detection of Nologging Operations On the Primary and Standby Databases

On the primary database, you can monitor for the most recent nologging operation that occurred in the database by issuing the following query:

```
SELECT NAME, UNRECOVERABLE_CHANGE#,
TO_CHAR (UNRECOVERABLE_TIME,'DD-MON-YYYY HH:MI:SS')
FROM V$DATAFILE;
```

The above primary database's query dictates when the most recent nologging operation occurred and when the invalidation redo was written to the redo.

Once Redo Apply (or Media Recovery) processes the invalidation redo, it marks all the corresponding data blocks corrupt. You will detect encounter corrupted blocks on the physical standby database when you query any data that references these data blocks. You will receive the following errors:

```
ORA-01578: ORACLE data block corrupted (file # 3, block # 514)
ORA-01110: data file 3: '/u01/lto_linux9206/dbs/users.dbf'
ORA-26040: Data block was loaded using the NOLOGGING option
```

You can proactively catch some of these corrupted blocks on Redo Apply (or media recovery) instance by running DBVERIFY on the data files.

```
$ dbv file=users.dbf
DBVERIFY - Verification starting : FILE = users.dbf
DBV-00200: Block, dba 12583426, already marked corrupted
DBV-00200: Block, dba 12583427, already marked corrupted
DBV-00200: Block, dba 12583428, already marked corrupted
```

SQL apply ignores the invalidation redo since it cannot convert it to any reasonable SQL; so, the logical standby will not receive any immediate errors. If future transactions reference the missing data, then apply slave will receive an ORA-01403 in the alert.log. For example, the following UPDATE statement failed on the logical standby because it was referencing 'nologged' rows that do not exist on the logical standby database.

```
LOGSTDBY stmt: update "SCOTT"."NOLOG"
  set
    "SAL" = 810
  where
    "EMPNO" = 7369 and
    "ENAME" = 'SMITH' and
    "JOB" = 'CLERK' and
    "MGR" = 7902 and
    "HIREDATE" = TO_DATE('17-DEC-80', 'DD-MON-RR') and
    "SAL" = 800 and
    "COMM" IS NULL and
    "DEPTNO" = 20 and
    ROWID = 'AAAAAAAAEAAAACRAAA'
LOGSTDBY status: ORA-01403: no data found
LOGSTDBY PID 21733, oracle@dlsun1917 (P004)
LOGSTDBY XID 0x0001.010.00000cf3, Thread 1, RBA 0x038b.00000826.1a4
Tue Nov  2 18:26:51 2004
Errors in file /private/oracle/app/admin/tens/bdump/tens_lsp0_20328.trc:
ORA-12801: error signaled in parallel query server P004
ORA-01403: no data found
LOGSTDBY Reader P003 pid=27 OS id=21729 stopped
```

Currently in Oracle 9i and Oracle 10gR1, only the primary database V$DATAFILE view reflects nologging operations.. In

10gR2, the V$DATAFILE view will be enhanced to include information regarding when an invalidation redo is applied and the aforementioned corrupted blocks are written to the corresponding data file on a Redo Apply (or media recovery or standby) instance.

Repair of Nologged Changes on the Physical and Logical Standby Databases

After a nologged operation on the primary is detected, it is recommended to create a backup immediately if you want to recover from this operation in the future. However there are additional steps required if you have an existing physical or logical standby database. This is crucial if you want to preserve the data integrity of your standby databases.

For a physical standby database, Redo Apply will process the invalidation redo and mark the corresponding data blocks corrupt.

For a physical standby database, follow these steps[2] to reinstantiate the relevant data files .

1. stop Redo Apply (recover managed standby database cancel)
2. offline corresponding datafile(s) (alter database datafile <NAME> offline drop;)
3. start Redo Apply (recover managed standby database disconnect)
4. copy the appropriate backup datafiles over from the primary database (e.g. use RMAN to backup datafiles and copy them)
5. stop Redo Apply (recover managed standby database cancel)
6. online corresponding data files (alter database datafile <NAME> online;)
7. start Redo Apply (recover managed standby database disconnect)

For a logical standby database, SQL Apply skips over the invalidation redo completely; so, the subsequent corresponding table or index will not be updated. However, future reference to missing data will result in ORA-1403 (no data found). In order to resynchronize the table with the primary table, you need to re-create it from the primary database. Follow the steps described in Oracle Data Guard Concepts and Administration, Chapter 'Managing a Logical Standby Database', and Section 'Adding or Re-Creating Tables On a Logical Standby Database' Basically, you will be using the DBMS_LOGSTDBY.INSTANTIATE_TABLE procedure.

[1]Invalidation redo containing information about the nologging operation and the range of blocks it affects.

[2]Please also refer to the Data Guard Concepts & Administration documentation.