

Making Oracle Streams Fly

Streams performance is largely dependant on your environment; the number of target sites you are replicating data to, the volume of data, the number of transactions per second occurring on the source database, the transformation logic, the hardware, the network etc, etc. However, in a simple downstream capture environment between a source and target database, it is possible to achieve very low latency even in heavy data load situations.

Archive log downstream capture

Consider a unidirectional archive log downstream capture configuration. In Oracle 11g, the capture and apply processes are combined. Here, the capture process acts as the propagation sender, transmitting Logical Change Records (LCRs) directly from the capture process to the apply process. In this mode, the buffered queue is optimised to improve efficiency.

Confirmation that Streams has adopted a combined capture and apply configuration can be seen in the target database instance alert log on Streams startup:

```
Streams CAPTURE CP01 for TGT_SCHEMA_CAPTURE with pid=43, OS id=27358
is in combined capture and apply mode.
Streams downstream capture TGT_SCHEMA_CAPTURE uses
downstream_real_time_mine: FALSE
Starting persistent Logminer Session with sid = 7 for Streams Capture
TGT_SCHEMA_CAPTURE
```

Even in this configuration, the data replication throughput and subsequent latency, is dependant on the source database archiving it's redologs. Oracle 11g has addressed this issue by providing a database initialisation parameter; `archive_lag_target`. Setting this on the source database (in seconds) allows redo log archiving to occur at defined intervals. Not too small to cause the source database to log file switch too frequently, and not too large to cause an increase in propagation latency.

In a recent test, I set the `archive_lag_target` parameter to 180 seconds, which provides a best effort minimum of 3 minutes of propagation latency. I.e. an archive log file will be shipped from the source database to the target database's foreign archive log destination every 3 minutes, regardless of data volumes. So, for an idle source database, Streams will be shipping relatively empty archive logs

To achieve an archive log downstream capture configuration as described, I set the following database parameters:

Source Database (SRC)

```
archive_lag_target = 180
```

```
log_archive_config = 'SEND, RECEIVE, DG_CONFIG=(SRC,TGT)'
```

```

log_archive_dest_1 = 'LOCATION=+FLASH
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=SRC'

log_archive_dest_2 = 'SERVICE=TGT ARCH NOREGISTER
VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE)
TEMPLATE=+FLASH/TGT/FOREIGN_ARCHIVELOG/SRC/src_arch_%t_%s_%r.l
og DB_UNIQUE_NAME=TGT'

log_archive_dest_state_1 = 'enable'

log_archive_dest_state_2 = 'enable'

```

Target Database (TGT)

```

log_archive_config = 'DG_CONFIG=(SRC,TGT)'

log_archive_dest_1 = 'LOCATION=+FLASH/'

log_archive_dest_state_1 = 'enable'

```

For the complete setup of Streams downstream capture, refer to the Oracle documentation:

http://download.oracle.com/docs/cd/B28359_01/server.111/b28321/strms_capture.htm#i1013269

Real-time downstream capture

To achieve a faster Streams downstream capture process realising minimum propagation latency use a “Real-time downstream capture” process. This method provides a “best of both worlds” approach, allowing a real-time mine configuration that falls back to archive log mining when the apply process cannot keep up. In addition, the real-time mine process is re-enabled automatically when data throughput is less.

Real-time downstream capture also utilises the Oracle 11g feature; combined capture and apply. However, it requires additional configuration to enable it, including the creation of standby redo logs on the target database that are directly written to by the source database’s log writer (LGWR).

The steps to convert Streams to Real-time downstream capture are described below:

1. Create $n+1$ standby redo logs on the target database (where n is the number of online redo logfiles per thread on the source database)

Target Database (TGT)

```

sqlplus '/as sysdba'
SQL> alter database add standby logfile size 1024M;

```



Tip: The size of each standby redo log file must be the same as that of the source database online redo log files. The example above is using Oracle Managed Files with ASM so no need to explicitly state the filename.

2. Stop the existing downstream capture process on the target database by executing the following procedure as streams admin user:

Target Database (TGT)

```
sqlplus streams_admin/streams_admin
```

```
SQL> exec
```

```
dbms_capture_adm.stop_capture(capture_name=>'SRC_SCHEMA_CAPTURE')
```

3. Set the following database initialisation parameters on the source database:

Source Database (SRC)

```
archive_lag_target = 0
```

```
log_archive_dest_2 = 'SERVICE=TGT ASYNC REOPEN=30 NOREGISTER  
VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE) DB_UNIQUE_NAME=TGT'
```

4. Set the following database initialisation parameters on the target database as streams admin user:

Target Database (TGT)

```
log_archive_dest_1 = 'LOCATION=+FLASH  
VALID_FOR=(ONLINE_LOGFILES, ALL_ROLES)'
```

```
log_archive_dest_2 =  
'LOCATION=+FLASH/TGT/foreign_archivelog/SRC  
VALID_FOR=(STANDBY_LOGFILES, ALL_ROLES)'
```

```
log_archive_dest_state_2 = 'enable'
```

5. Set the downstream real-time mine Streams capture initialisation parameter to 'Y' on the target database by executing the following procedure as streams admin user:

```
SQL>
```

```
BEGIN
```

```
    dbms_capture_adm.set_parameter( capture_name =>  
'SRC_SCHEMA_CAPTURE',  
                                   parameter      =>  
'downstream_real_time_mine',  
                                   VALUE          => 'Y');
```

```
END;
```

```
/
```

6. Start the downstream capture process on the target database by executing the following procedure as streams admin user:

```
SQL> exec
dbms_capture_adm.start_capture(capture_name=>'SRC_SCHEMA_CAPTURE')
```

7. Log back onto source database as SYSDBA and switch log files to initiate the real-time downstream capture process.

Source Database (SRC)

```
sqlplus / as sysdba
```

```
SQL> alter system archive log current;
```

8. Logon to target database as streams admin user and check capture process is “capturing changes”.

```
SQL> select CAPTURE_NAME, STATE from v$streams_capture;
```

CAPTURE_NAME	STATE
-----	-----
SRC_SCHEMA_CAPTURE	CAPTURING CHANGES

Alert log

Confirmation that Streams is using Real-time downstream capture can be seen in the target database instance alert log upon Streams startup:

```
Streams CAPTURE CP01 for SRC_SCHEMA_CAPTURE with pid=106, OS id=7896
is in combined capture and apply mode.
Streams downstream capture SRC_SCHEMA_CAPTURE uses
downstream_real_time_mine: TRUE
Starting persistent Logminer Session with sid = 38 for Streams
Capture SRC_SCHEMA_CAPTURE
```

Also note the transition to mine standby redo logs (Realtime mine) for thread 1 sequence 11

```
<snip>
RFS[17]: Selected log 20 for thread 1 sequence 11 dbid 2053587448 branch
708614321
LOGMINER: End mining logfile for session 38 thread 1 sequence 10,
+FLASH/tgt/foreign_archivelog/src/tgt_1_10_708614321.dbf
LOGMINER: Begin mining logfile for session 38 thread 1 sequence 11,
+FLASH/tgt/onlineolog/group_20.6324.704635273 ← HERE
</snip>
```

Tuning the Capture process

So you have successfully configured Real-time downstream capture, but this is not the whole story regarding Streams performance tuning, just a step in the right direction. Let's take a look at the hardware specification of the target database (where Streams capture process is running) and tune some Capture process parameters accordingly.

Hardware specification

HP ProLiant DL585 G2 Server
4 x 2 Core CPUs
32 GB RAM
EMC DMX-4 SAN Storage Array

Having spent many weeks evaluating numerous configurations, the following Capture process initialisation parameters provided the best performance for a 1000 transaction per second load (tps), equating to approximately 7000 LCRs per second:

Streams Capture Parameters

DOWNSTREAM_REAL_TIME_MINE	Y
PARALLELISM	2
_CHECKPOINT_FREQUENCY	1000
_SGA_SIZE	100
DML Handler	No
Error Handler	Yes

1. Add capture performance settings using `DBMS_CAPTURE_ADM.SET_PARAMETER` and `DBMS_CAPTURE_ADM.ALTER_CAPTURE` procedures, as in example below:

```
BEGIN
  dbms_capture_adm.set_parameter( capture_name =>
'SRC_SCHEMA_CAPTURE',
                                parameter   =>
'_CHECKPOINT_FREQUENCY',
                                VALUE       => '1000');
  dbms_capture_adm.alter_capture( capture_name =>
'SRC_SCHEMA_CAPTURE',
                                checkpoint_retention_time => 7);
  dbms_capture_adm.set_parameter( capture_name =>
'SRC_SCHEMA_CAPTURE',
                                parameter   => 'PARALLELISM',
                                VALUE       => '2');
  dbms_capture_adm.set_parameter( capture_name =>
'SRC_SCHEMA_CAPTURE',
                                parameter   => '_SGA_SIZE',
                                VALUE       => '100');
END;
/
```



Tip: Streams will stop and start the Capture process automatically.

DB Initialisation Parameters

streams_pool_size (minimum)	4G
memory_target	24G

Tuning the Apply process

Tuning the Apply process is a similar approach. Based on the same hardware, the best performing configuration for schema replication given a 1000 tps (7000 LCRs per second) workload is as follows:

Streams Apply Parameters

```
COMMIT_SERIALIZATION  NONE
PARALLELISM           32
TXN_LCR_SPILL_THRESHOLD 1000000
_HASH_TABLE_SIZE      20000000
_KGL_CACHE_SIZE       100
_TXN_BUFFER_SIZE      960
```

1. Add apply process performance settings using

DBMS_APPLY_ADM.SET_PARAMETER and DBMS_APPLY_ADM.ALTER_CAPTURE procedures, as in example below:

```
BEGIN
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  => 'disable_on_error',
                                VALUE      => 'n');
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  => '_HASH_TABLE_SIZE',
                                VALUE      => '20000000');
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  => 'parallelism',
                                VALUE      => '&app_para');
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  => '_DYNAMIC_STMTS',
                                VALUE      => 'Y');
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  =>
'TXN_LCR_SPILL_THRESHOLD',
                                VALUE      => '1000000');
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  => '_txn_buffer_size',
                                VALUE      => '960');
  dbms_apply_adm.set_parameter( apply_name => 'SRC_SCHEMA_APPLY',
                                parameter  =>
'commit_serialization',
                                VALUE      => 'NONE');
END;
/
```



Tip: Streams will stop and start the Apply process automatically.

Measuring Streams Performance

For Oracle 11gR1, STRMMON functionality has been replaced by the Streams Performance Advisor distributed within the Streams product and implemented through DBMS_STREAMS_ADVISOR_ADM PL/SQL package.
Metalink Note: 732644.1 describes how to use the package in greater detail.

For my performance tests I found the DBMS_STREAMS_ADVISOR_ADM very useful for providing all the necessary information and statistics. The package contains just 1 procedure; ANALYZE_CURRENT_PERFORMANCE that takes a "snapshot" of the current Streams environment, a similar concept to AWR snaps.

N.B. Performance statistics require a comparison between 2 Streams Advisor snapshots.

The script below provides a complete Streams performance report, calling the DBMS_STREAMS_ADVISOR_ADM.ANALYZE_CURRENT_PERFORMANCE procedure and executing a number of queries against the TP views that are populated by the procedure. The TP views are "Temporary Performance" Streams data dictionary views containing performance and session related statistics.

```
#!/bin/sh
#
# run_streams_advisor.sh
#
. ~/.bash_profile

sqlplus streams_admin/streams_admin <<EOF
exec DBMS_STREAMS_ADVISOR_ADM.ANALYZE_CURRENT_PERFORMANCE;
alter session set nls_date_format='DD/MM/YYYY HH24:MI';
set lines 1000 pages 1000
set colsep '|'

COLUMN PATH_ID HEADING 'Path ID' FORMAT 999
COLUMN COMPONENT_ID HEADING 'Component ID' FORMAT 999
COLUMN COMPONENT_NAME HEADING 'Name' FORMAT A40
COLUMN COMPONENT_TYPE HEADING 'Type' FORMAT A12
COLUMN STATISTIC_NAME HEADING 'Statistic' FORMAT A32
COLUMN STATISTIC_VALUE HEADING 'Value' FORMAT 99999999.99
COLUMN STATISTIC_UNIT HEADING 'Unit' FORMAT A30
COLUMN ADVISOR_RUN_REASON FORMAT A25
COLUMN BOTTLENECK_IDENTIFIED FORMAT A25
COLUMN COMPONENT_DB FORMAT A20
COLUMN SOURCE_COMPONENT_DB FORMAT A30
COLUMN SOURCE_COMPONENT_NAME FORMAT A40
COLUMN DESTINATION_COMPONENT_DB FORMAT A20
COLUMN DESTINATION_COMPONENT_NAME FORMAT A40
COLUMN GLOBAL_NAME format a25
COLUMN VERSION format a10
COLUMN COMPATIBILITY format a15

set trimspool on
spool /home/oracle/streams/streams_advisor/streams_advisor.txt append
select * from DBA_STREAMS_TP_COMPONENT;
```

```
select * from DBA_STREAMS_TP_COMPONENT_LINK;
select * from DBA_STREAMS_TP_COMPONENT_STAT;
select * from DBA_STREAMS_TP_DATABASE;
select * from DBA_STREAMS_TP_PATH_BOTTLENECK;
select * from DBA_STREAMS_TP_PATH_STAT;
spool off
EOF
```



Tip: The output spool file `streams_advisor.txt` can be imported into MS Excel as a pipe '|' delimited file for further analysis and trend reporting.

During my tests, I ran the `run_streams_advisor.sh` via linux crontab job at 5 minute intervals.

Conclusion

With an optimal Streams configuration, it is still possible that the capture process will not stay in Real-time mine, i.e. when the Apply process can't keep up with the demand and the Capture process pauses in flow control, causing Streams to start mining the archived logs until it can "catchup". This scenario "tips the scales" in terms of latency (the time taken for a transaction to be replicated on the target database).

During my Streams schema replication performance testing I was able to maintain a 7 second latency at 1000 tps workload. Anything above this would cause a linear growth in latency that would quickly recover (back to Real-time mine) once the workload reduced to 200 tps.