Oracle Streams is simple in principle, but has lots of "moving parts." The initial setup may look easy, but don't be fooled! There are numerous pitfalls, functional as well as performance, that can easily trap the unsuspecting DBA, and cause massive performance degradation.

In this column we'll review the most critical factors that can wreck Streams performance. To get started, let's review the main components in Streams. (Note that our recommendations are applicable to Oracle 10g.)

OVERVIEW OF STREAMS ARCHITECTURE

The three Streams components are called *Capture*, *Propagate*, and *Apply*. The capture process is simple in concept. Using LogMiner, Streams continually reads the redo log, checks if the transaction is part of the propagation rule set, and then stores the change on a queue in memory. These steps are all very fast, meaning that capture will seldom be a performance bottleneck. The second step, propagation, is also not usually a problem, as long as the network is fast.

In contrast to the first two steps, the apply process bears the brunt of the load. The apply processes must actually perform the transaction. This typically involves at least some disk i/o, and possibly much cpu. Streams employs a clever way to protect the apply process from overload. When the apply process falls behind, the capture process throttles itself back, temporarily stopping capture. This state is called "Paused for Flow Control." This moderating effect is a smart move, because the apply process runs badly when overwhelmed. If Streams allowed the destination queue to fill-up, the apply process would be forced to "spill" the excess transactions out of the queue, and then re-read them later.

Let's now investigate some of the critical performance factors.

NUMBER OF APPLY PROCESSES

On busy OLTP systems, with lots of users doing lots of transactions on the source database, you will need a commensurate number of apply processes at the destination. A reasonable starting point is to use the same number of apply processes as you have cpus at the destination site. If the extra processes remain idle most of the time, no harm is done.

The number of apply processes is controlled via a special apply parameter called ***Parallelism***. Note that an apply parameter is not the same as an init.ora parameter. To set an apply parameter you use the procedure *Dbms_Apply_Adm.Set_Parameter*.

COMMIT FREQUENCY

This is the single most important area to consider when working with Streams. Here's why: Outside Streams, with regular batch jobs, the commit frequency used is not catastrophic, as long as one doesn't commit too often. So, if you are running an insert of one million rows, you can commit every 1,000 rows, every 10,000 rows or maybe wait until the very end, as long as your undo space is adequate. We normally just want to avoid excessive commits—(especially once per row).

With Streams, however, it is critical that you commit at moderate intervals—roughly every 1,000 rows. If you don't commit at regular intervals, you will pay a double penalty. Firstly, your

propagated data (called Logical Change Records, or LCRs) may overflow the buffered queue at the destination database. Copying LCRs back and forth causes large performance delays. Secondly, infrequent commits means that Streams cannot take advantage of multiple apply processes. This is due to a Streams restriction that a single transaction must be handled by only a *single* process.

COMMIT_SERIALIZATION.
Using this apply parameter, Streams allows you to control the commit order in the destination database. You can preserve the same order in which transactions committed in the original database (parameter set to ***full***), or allow variation (parameter set to ***none***).

Allowing unrestricted commit order improves performance somewhat because the apply processes never need to wait for any earlier transactions to commit. Nevertheless, the ***none*** setting should only be used if you are positive that changing the commit order will not cause problems in your application.

CHECKPOINT FREQUENCY
A Streams checkpoint is not the same as a database checkpoint. Instead, this refers to how often the Log Miner process stores its current processing status in the table, *System.Logmnr_Restart_Ckpt$.* You might think that this checkpoint data should not affect Streams performance, but that is not the case. This table is actively used, with *Selects*, *Inserts*, and *Updates* occurring very often. Further, on a busy database, each checkpoint may contain several thousand rows, with each row containing two *blob* fields (plus other fields, such as SCN#).

By default, Streams writes checkpoints every 10 Mb of transaction data, but this is changeable using the *capture* parameter *_Checkpoint_Frequency*. In practice, 10 Mb is much too low for busy systems, and causes the checkpoint table to quickly grow. A more appropriate value is 200. Set this checkpointing parameter using the procedure, *Dbms_Capture_Adm.Set_Parameter*. You can list the capture parameters by querying *Dba_Capture_Parameters*.

CHECKPOINT RETENTION
A related parameter than should be changed is *Checkpoint_Retention_Time*. This parameter controls how long checkpoint data is saved**.** In practice, the default value of 60 (days) is excessive; a better value is just 2.

An interesting quirk about this parameter is that you set it a bit differently than the other parameters. *Checkpoint_Retention* is set when a capture process is first defined. If you need to change it later, use the procedure, *Dbms_Capture_Adm.Alter_Capture.*

OTHER CRITICAL PARAMETERS
There are a few other key parameters that need to be changed:

- **_Hash_Table_Size**: This is another *apply* parameter. Recommendation from Oracle support is to use 10000000.

- **Aq_Tm_Processes**: This should be set to 1. (Note that the 10.2.0.3 *Upgrade Assistant* incorrectly changes this to 0!)

- **Txn_Lcr_Spill_Threshold**: This determines the maximum number of LCRs in a single transaction, after which the apply queue overflows. To avoid this, set this parameter slightly larger than the estimated maximum number of LCRs in any single transaction.

Note that you can check your current Streams parameters by querying the views, *Dba_Apply_Parameters* and *Dba_Capture_Parameters*.

CHECKING STREAMS PERFORMANCE

Finally, if you want a quick snapshot of how well Streams is working, Oracle provides an easy way. Here's my favorite Streams script for measuring the end-to-end response time. In the (real) example below, Streams captured, propagated, and applied the most recent transaction in 1 second.

```
COLUMN  APPLY_PROC FORMAT A12
COLUMN  LAT_SEC FORMAT 999999999
COLUMN 'Message Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN MSG_NO FORMAT 9999999999999

 SELECT APPLY_NAME APPLY_PROC,
(HWM_TIME-HWM_MESSAGE_CREATE_TIME)*86400 LAT_SEC,
TO_CHAR(HWM_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
"Message Creation",
TO_CHAR(HWM_TIME,'HH24:MI:SS MM/DD/YY') "Apply Time",
HWM_MESSAGE_NUMBER MSG_NO
FROM V$STREAMS_APPLY_COORDINATOR;
```

```
APPLY_PROC      Lat Sec Message Creation  Apply Time                MSG_NO
------------ ---------- ----------------- ----------------- --------------
APP_PRD               1 10:39:43 04/25/07 10:39:44 04/25/07  3316562531369
```

SUMMARY: POWERFUL BUT TRICKY

Streams can be a great way to propagate information, but there are many complications to consider. In concept, Streams is simple, but efficient implementation takes great care. With a little foresight, and by following these basic suggestions, you can avoid the most serious pitfalls.