ORACLE

All About Oracle Sequences

Chinmayi Krishnappa
Consulting Member of Technical Staff
DST, Oracle

Background

- What?
  - Oracle sequences are unique number generators that follow a user defined order (ascending/descending, step, cycle, etc)
- Why?
  - Introduced in ver 6
  - Application workarounds require higher level serialization
  - Many applications simply require that identifiers are unique
- Tradeoffs
  - Does not guarantee commit time ordering
  - Gaps in the case of rollback, process, instance, or system failure
  - Loss of global ordering in RAC settings
- GUARANTEE : unique values!

Syntax

- Create sequence <name> <attributes>;
  - Cache <n>|nocache, order|noorder,
  - keep|nokeep, session|global,
  - cycle <n>, start with <n>, increment by <n>, minvalue <n>, maxvalue <n>
  - scale|noscale <extend|noextend>
- Alter sequence <name> <attributes>;
  - Restart <start with <n>
- NEXTVAL and CURRVAL
  - Select <sequence_name>.nextval from dual;
  - Select <sequence_name>.currval from dual;

Usage

- Typically used to generate automatic primary keys for tables
- Ordering (e.g. ids for a reservation system)
- Generate unique number

- CREATE TABLE t1 (id NUMBER GENERATED **[ALWAYS|BY DEFAULT|BY DEFAULT ON NULL]** AS IDENTITY);
- Pre 12g: Create a sequence, create a BEFORE INSERT trigger, and call the NEXTVAL value of the sequence within the trigger

KEEP attribute and dbms_shared_pool.keep

- KEEP|NOKEEP
  - Relates to Application Continuity (12.1)
  - KEEP generate the same sequence value during replay.
  - Default: NOKEEP
- DBMS_SHARED_POOL.KEEP
  - Keeps the sequence object pinned in the shared pool
  - Useful for hot sequence objects when
    - Shared pool is not appropriately sized, and/or
    - Busy workload that ages out objects frequently

Sequence Cache

- Cached (CACHE <n>) and Uncached (NOCACHE) sequences
- Default: cached, with a cache size of 20
- Cache is instance local and ordered across sessions
- Reduces round trips to disk

- **Tradeoff**: can lose a cache of sequence numbers with session/instance failures

Ordered and Unordered sequences

- Sequences are unordered by default
- Ordered sequences (ORDER) enforce strict ordering of values across sessions
- Use of ordered sequence on RAC is discouraged
  - Requires a DLM lock for every nextval
- Ordered sequences are almost always cached
  - How does ordering work with instance local caches?

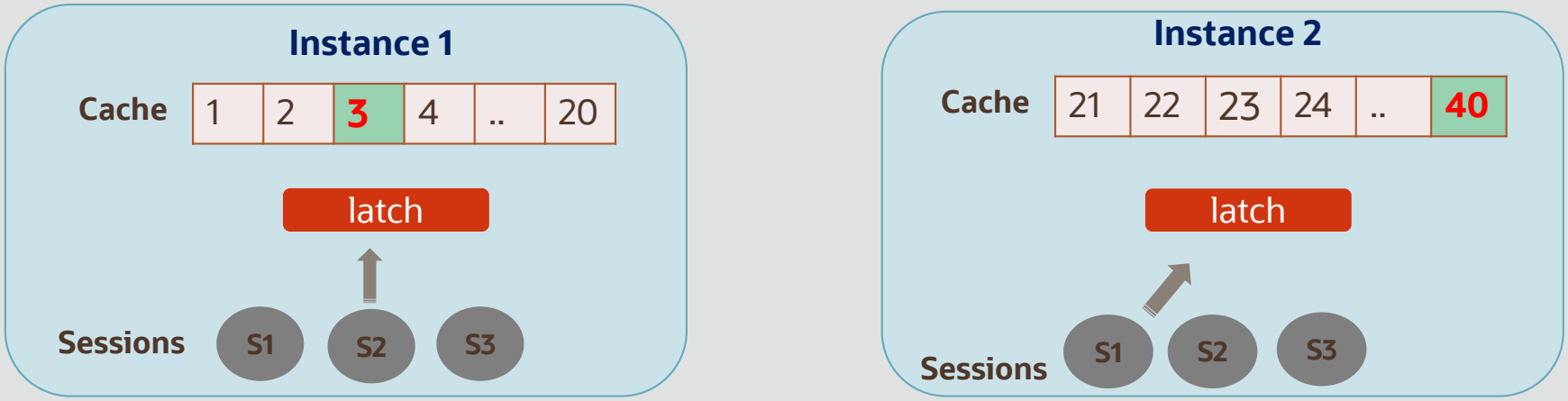
Synchronization points

- Latch (sequence cache)
  - Protects access to the instance local cache
- Enqueue (SQ) for cache replenishment
  - Protects sequence row cache (dc_sequences)
  - Increase cache size
- SV (ordering)
  - Alter sequence to be unordered
  - Increase cache size (something SQ contention shows up as SV)

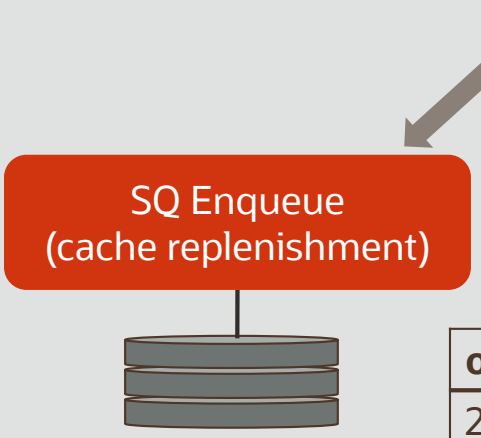
Recent performance bottlenecks

- Customer 1
  - Moving from a single instance to RAC
  - 2-node active-passive RAC cluster with ordered sequence primarily used on one instance
  - Average SQ enqueue wait time of 2 seconds
- Customer 2
  - 2-node RAC on 11.2.0.4, moving to X8 Exacs
  - Cannot get rid of ordered sequence for historic reasons
  - Average SV enqueue wait time shot up from 5 ms to 150 ms

Unordered sequences



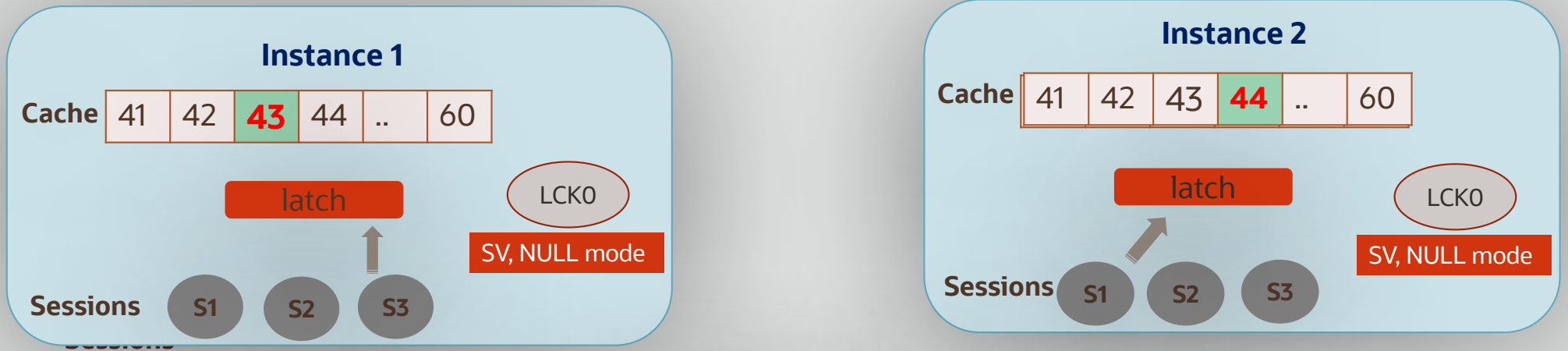
1. Latch serializes access to instance local cache
2. If cache depleted, get the SQ enqueue to replenish the cache



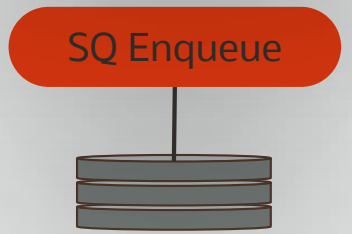
DC_SEQUENCES

obj#	cache	highwatermark
2103	20	60

Ordered sequences share the cache



1. Get SV enqueue
2. Read nextval from lock value
3. Adjust instance cache such that nextval falls within cache boundaries
4. If cache depleted, replenish under SQ
5. Write new nextval to lock value
6. Release SV enqueue



DC_SEQUENCES

obj#	cache	highwatermark
2103	20	60

Planned

- Dynamic cache resizing (21)
  - Elastically grow/shrink the cache based on usage
- Future work
  - Ideas to reduce SV contention

Scalable Sequences

Scalable Sequences - Motivation

- Index on column populated using a sequence generator results in a right growing index
- Last leaf block is a hot spot leading to contention

- Traditional ways to reduce contention:
  - A large CACHE value results in statistical affinity of index leaf blocks to instances
  - A reverse key index exchanges a single hot block for many cold blocks
  - A global index partitioned by hash reduces contention

Scalable Sequences - Motivation

- Limitations of prior approaches
  - A large CACHE value does not reduce index leaf block contention on single instance and SMP systems
  - A reverse key index usually reduces contention but at the expensive of a dramatic increase in expensive physical reads and writes
  - A global index partitioned by hash does not improve affinity of index leaf blocks to instances

Scalable Sequences

- Ideal Solution
  - Reduced contention without the penalty of significant increases in physical reads and writes
  - Affinity of index leaf blocks to instances for a Real Application Clusters database
  - Affinity of index leaf blocks to processes for an SMP system
  - No requirement for application modifications

Scalable sequence - Internals

- A numeric offset is prefixed to nextval
  - iii||sss, where
  - iii => (instance_id % 100) + 100
  - sss => by (session_id % 1000)

- The most significant “1” in the prefix prevents duplicates
  - 100||100||12, where instance offset=100, session offset=100.
  - 100||000||10012 , where instance offset=100, session offset=000.
  - Without the leading “1”, these values are essentially duplicates.

Scalable Sequences – NOEXTEND

- NOEXTEND (default)
  - values have same number of digits as maxvalue/minvalue
  - useful for integration with existing applications
  - If maxvalue = 1000000, then nextvals are iii||sss||1, iii||sss||2, … , iii||sss||9, followed by an error

Scalable Sequences – EXTEND

- NOEXTEND (default)
  - values have same number of digits as maxvalue/minvalue
  - useful for integration with existing applications
  - If maxvalue = 1000000, then nextvals are iii||sss||1, iii||sss||2, … , iii||sss||9, followed by an error
- EXTEND
  - Nextvals are all of length (x+y), where x is the length of the scalable offset (6), and y is the length of maxvalue/minvalue.
  - If maxvalue=100, then nextvals are of the form iii||sss||001, iii||sss||002, …,iii||sss||100

Scalable sequences -Syntax

- CREATE/ALTER sequence
  - *{ SCALE {EXTEND | NOEXTEND} | NOSCALE}*
  - Create sequence s1 scale; //noextend
  - Alter sequence s1 scale extend;
- NOSCALE disables sequence scalability
  - If any scaled values were returned, then we run the risk of generating a duplicate
  - Solution: Hwm on disk is prefixed with the largest possible prefix (199999)
- Not recommended with ordered sequences

Scalable sequence - examples

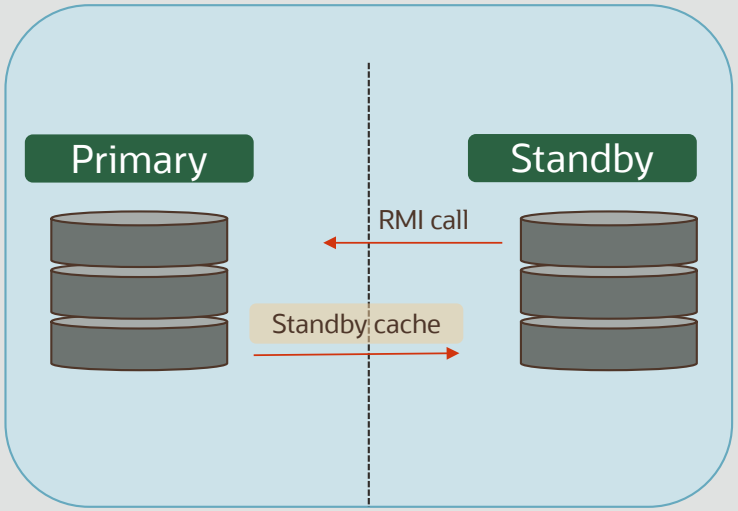
- Maxvalue = 1000000, increment = 1
- Instance offset 123, session offset 789

- scale noextend:
  - 11237891, 11237892, …,11237899 <error>

- scale extend:
  - 1123789000001, 112378900000002, …1123789000009,….,

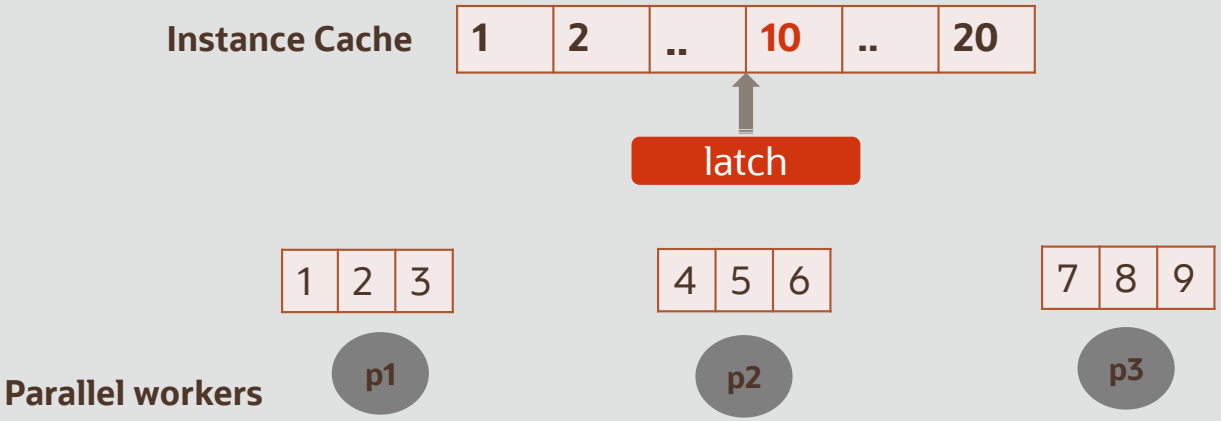
Sequences on ADG

- Standby never updates hwm on disk
- Sequence on standby always gets a cache of values from the primary.



Parallel dml

- update /*+ enable_parallel_dml parallel(3) */ table1 set id = seq1.nextval;



Session Sequence & Restart

- Session sequence numbers are local to session
- Create sequence seq1 session;
- Automatic restart for every new session
  - 1, 2, 3, 4, <new session>, 1, 2, 3, 4

- RESTART (18c)
  - 1, 2, 3, 4, 5
  - Alter sequence seq1 **RESTART** [**START WITH** 4];
  - 4, 5, 6, 7

Monitoring Sequence Usage

- user_sequences, all_sequences, dba_sequences
  - scale, extend flags
- v$_sequences
  - cache_size, nextvalue, order_flag, highwater
- Tracing
  - event="10290 trace name context forever, level <1-5>"

Questions?