

Systemstate dump analysis: Nocache on high intensive sequences in RAC

Sequence values are cached in instance memory and by default 20 values are cached. As instance cache is transient, loss of an instance can result in loss of cached sequence values. Permanent record of highest possible value from any instance is kept track in SEQ\$ table.

SQLs accessing this sequence within an instance will access instance SGA. As each instance caches its own sequence values it is highly likely that SQLs accessing this sequence from different instance will create gaps in sequence values.

Common knee-jerk reaction to this issue is to set nocache or cache 1 for these sequences. In a single instance environment, this approach will backfire due to massive updates to SEQ\$ tables, buffer busy waits, latch free waits etc. In a RAC environment, this issue is magnified and almost hangs the instance. I had the privilege of working with a client to resolve one of their performance issues.

Problem

Instances were hung. It was not possible to login to the database. Many existing connections are working fine though. We were lucky enough that one of the DBAs had active connection to the database. So, we took systemstate dump from that session to see why there is a hang (or slowness).

Alert log was printing following lines at this time frame. So, we know that there is a problem with row cache enqueue.

```
Wed Feb 18 06:31:31 2008
>>> WAITED TOO LONG FOR A ROW CACHE ENQUEUE LOCK! pid=45
Wed Feb 18 08:59:31 2008
>>> WAITED TOO LONG FOR A ROW CACHE ENQUEUE LOCK! pid=44
Wed Feb 18 08:59:31 2008
>>> WAITED TOO LONG FOR A ROW CACHE ENQUEUE LOCK! pid=43
Wed Feb 18 09:01:00 2008
>>> WAITED TOO LONG FOR A ROW CACHE ENQUEUE LOCK! pid=46
Wed Feb 18 09:01:00 2008
>>> WAITED TOO LONG FOR A ROW CACHE ENQUEUE LOCK! pid=27
```

systemstate dump

We took a systemstate dump with following command.

```
alter session set events 'immediate trace name systemstate level 4';
```

I won't bore you with all the details. But let us review just necessary sections of the systemstate dump file. We know that processes were waiting for row cache enqueue and so I searched for row cache enqueue in the trace file. From the trace file, we see that many sessions are holding NULL mode row cache enqueue and requesting X mode.


```

user=6136d45b8 session=6136d45b8 count=1 flags=[00] savepoint=58
LIBRARY OBJECT HANDLE: handle=64cdf5868
name=SELECT SCOTT.STG__SEQ.NEXTVAL FROM DUAL
hash=5dbaf014 timestamp=02-18-2008 08:00:46
namespace=CRSR flags=RON/KGHP/TIM/PN0/SML/[12010000]
kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch#=20
lwt=64cdf5898[64cdf5898,64cdf5898] ltm=64cdf58a8[64cdf58a8,64cdf58a8]
pwt=64cdf58c8[64cdf58c8,64cdf58c8] ptm=64cdf5958[64cdf5958,64cdf5958]
ref=64cdf5878[64cdf5878, 64cdf5878]
lnd=64cdf5970[64cdb02f8,64ce05748]
LIBRARY OBJECT: object=64cdf4b40
type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0

```

Sequence cache

In summary, we know that sessions were waiting for row cache enqueues with QN type. That row cache enqueue was protecting a parent row cache entry for dc_sequences. SO [State Objects] for both sequences and cursors accessing that sequence is also exists in the systemstate dump file.

Let's query properties of that sequence.

```
SQL> select * from dba_sequences where sequence_name='STG__SEQ';
```

SEQUENCE_OWNER	SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	C	O	CACHE_SIZE	LAST_NUMBER
SCOTT	STG__SEQ	1.0000E+27		1	N	N	0	338585872

Evidently cache size is set to 0. High last_number also indicates that this is an heavily accessed sequence.

Summary

Essentially, keeping sequence with nocache in RAC for highly accessed sequences is pretty bad idea. Login to the database accesses sequences too, so if row cache enqueues are not available, then logins will hang. This is why existing connections were working fine, but new logins were simply hung. We are able to pinpoint all these using systemstate dump.