

Resolving HW enqueue contention
Riyaj Shamsudeen
orainternals.wordpress.com

Recently, I had few email exchanges about HW enqueue contention in oracle-1 list and offline. There were few interesting observations from the test case.

HW enqueue

When a session needs access to a resource, it requests a lock on that resource in a specific mode. Internally, lock structures are used to control access to a resource. Enqueues, as name suggests, have First In First Out queueing mechanism.

Segments have High Water Mark (HWM) indicating that blocks below that HWM have been formatted. New tables or truncated tables (truncated without reuse storage clause), have HWM value set to segment header block. Meaning, there are zero blocks below HWM. As new rows inserted or existing rows updated (increasing row length), more blocks added to the free lists and HWM bumped up to reflect these new blocks. HW enqueues are acquired in Exclusive mode before updating HWM and HW enqueues operate as a serializing mechanism.

In non-ASSM tablespaces, HWM is bumped up by 5 blocks at a time (Actually, undocumented parameter `_bump_highwater_mark_count` controls this behavior and defaults to 5). Heavy inserts in to a table can result in increased HWM activity leading to HW enqueue contention. This issue is prevalent if the table has LOB columns or if the row length is big.

Measuring HW enqueue contention

We will use few test cases to see how underlying extent size and table structures are affecting HW enqueue contention. But, before we need to find a way to measure total number of gets on HW enqueue. If total number of gets on HW enqueue is reduced, enqueue contention can be relieved.

Fixed table `x$ksqst` stores statistics about total number of enqueue gets, success and failures of those gets at instance level. For example, to see total number of gets on HW enqueue, following query can be used. `ksqstreq` indicates total # of gets and `ksqstwat` shows total # of waits.

```
SQL> select ksqstreq, ksqstwat from x$ksqst where ksqsttyp='HW';
```

KSQSTREQ	KSQSTWAT
546953	50

From Oracle version 10g and above, `x$ksqst` is externalized as `v$enqueue_statistics`.

But, this is at instance level. While we can use this statistics to measure activity on HW enqueue, we need to make sure that there is no other session acquiring HW enqueue. Event 10704 can be used for this and every call to get an enqueue prints few lines in the trace file. SQL statement to dump this information to trace file is :

```
alter session set events '10704 trace name context forever, level 15';
```

Event 10704 is documented as below:

```
10704, 00000, "Print out information about what enqueues are being obtained"
// *Cause: when enabled, prints out arguments to calls to ksqcmi and
// ksqlr1 and the return values.
// *Action: Level indicates details:
// Level: 1-4: print out basic info for ksqlr1, ksqcmi
// 5-9: also print out stuff in callbacks: ksqlac, ksqlop
// 10+: also print out time for each line
```

Few lines from the trace files printed below. ksq is internal Oracle module names for enqueues and ksqgtl is to get locks on a resource. From the lines below, we can see that HW enqueue is acquired in mode 6, exclusive mode. Timestamp is also printed since we enabled this event at level 15.

```
*** 2008-05-04 10:08:35.734
ksqgtl *** HW-00000007-01800014 mode=6 flags=0x11 timeout=21474836 ***
ksqgtl: xcb=0x1E283158, ktcdir=2147483647, topxcb=0x1E283158
        ktcipr(topxcb)=0x0

*** 2008-05-04 10:08:35.734
ksucti: init session DID from txn DID:
ksqgtl:
        ksqlkdid: 0001-0014-00000016

*** 2008-05-04 10:08:35.734
*** ksudidTrace: ksqgtl
        ktcmydid(): 0001-0014-00000016
        ksusesdi: 0000-0000-00000000
        ksusetxn: 0001-0014-00000016
ksqgtl: RETURNS 0

*** 2008-05-04 10:08:35.750
ksqrc1: HW,7,1800014
ksqrc1: returns 0
```

Now, we can grep for HW- in the trace file, count it and match that against v\$enqueue_statistics. Following test case illustrates this using an example table.

```
SQL> select ksqstreq, ksqstwat from x$ksqst where ksqsttyp='HW';
```

```
-----
 KSQLSTREQ  KSQLSTWAT
-----
      546198          50
```

```
SQL> insert into test_hw
select n, lpad(n, 4000,'a') v1 from
(select level n from dual connect by level <1000);
```

999 rows created.

```
SQL> SQL> SQL> commit;
```

Commit complete.

```
SQL> select ksqstreq, ksqstwat from x$ksqst where ksqsttyp='HW';
```

```
-----
 KSQLSTREQ  KSQLSTWAT
-----
      546953          50
```

A difference of 755 (546953-546198) total gets to HW enqueue.

Searching in the trace file for HW- enqueues in the trace file also prints 755.

```
/oracle/app/oracle/admin/TEST1/udump> grep 'HW-' test1_ora_26668.trc |wc -l
755
```

So, in my test database, my session is the only session and we could query v\$enqueue_statistics.

Test case

Following test case will be used to see the impact of extent size, tablespace management on HW enqueue gets. If total number of gets to HW enqueue can be reduced, contention can be relieved.

Following script creates a tablespace, creates a table in that tablespace, inserts 9999 rows in to the table and prints total enqueue gets.

```
-- script enq_hw1.sql -----
variable v_begin_cnt number
variable v_end_cnt number

prompt
prompt Tablespace: Locally managed with \&1
prompt
drop tablespace TS_LMT_HW including contents and datafiles;

create tablespace TS_LMT_HW datafile 'D:\ORACLE\ORADATA\ORCL11G\TS_LMT_AUTO_1M_01.DBF'
size 200M
extent management local
\&1;

create table test_hw (n1 number , c1 clob ) tablespace TS_LMT_HW;

begin
select total_req# into :v_begin_cnt from v$enqueue_statistics where eq_type = 'HW';
end;
/
insert into test_hw
select n, lpad(n, 4000, 'a') v1 from
(select level n from dual connect by level <10000);
commit;
select total_req#, succ_req#, failed_req# from v$enqueue_statistics where eq_type = 'HW';

begin
select total_req# into :v_end_cnt from v$enqueue_statistics where eq_type = 'HW';
end;
/

select :v_end_cnt - :v_begin_cnt diff from dual;

--- script end enq_hw1.sql -----
```

Above script is called by following script passing various tablespace attributes:

```
spool call_eng_hw1.lst
@enq_hw1 "uniform size 5M segment space management manual"
@enq_hw1 "uniform size 5M segment space management auto"
@enq_hw1 "uniform size 40K segment space management manual"
@enq_hw1 "uniform size 40K segment space management auto"
@enq_hw1 "autoallocate segment space management manual"
@enq_hw1 "autoallocate segment space management auto"
spool off
```

Test Results

Tested above script for tablespace with various attributes in versions 10.2.0.4 and 11.1.0.6.

Test	Extent management	Segment space management	10.2.0.4	11.1.0.6
#1	Uniform 5M	Manual	2049	64
#2	Uniform 5M	Auto	50	48
#3	Uniform 40K	Manual	6604	6045
#4	Uniform 40K	Auto	7552	7554
#5	Autoallocate	Manual	2231	279
#6	Autoallocate	Auto	269	291

There are few key points here.

1. If tablespace uniform size is too low, then # of HW enqueue gets increases sharply. Compare test cases #1 & #3. Enqueue gets decreased 10 times, when tablespace extent size is set properly.

2. In test case #1 and #5, there is a dramatic decrease in enqueue gets between 10g and 11g. Looks like, a new feature faster lob has been introduced. Tested above script for a table without lob column. Virtually there is no difference between 10g and 11g, if there is no lob column.

```

create table test_hw (n1 number , v1 varchar2(1000), v2 varchar2(1000), v3 varchar2(1000) )
tablespace TS_LMT_HW;
...
declare
i number;
begin
for i in 1 .. 10000
loop
insert into test_hw values ( i, lpad(i, 1000, 'a'), lpad(i, 1000, 'a'), lpad(i, 1000, 'a') );
commit;
end loop;
end;
/
...

```

Test	Extent management	Segment space management	10.2.0.4	11.1.0.6
#1	Uniform 5M	Manual	1020	1019
#2	Uniform 5M	Auto	33	31
#3	Uniform 40K	Manual	3004	3004
#4	Uniform 40K	Auto	3055	3055
#5	Autoallocate	Manual	1132	1132
#6	Autoallocate	Auto	163	163

3. In all test cases, automatic segment space management tablespaces performed fewer enqueue gets. In ASSM tablespaces HWM is bumped up by much higher number.

4. Allocating additional extent with instance keyword seems to help in non-ASSM tablespace. It is different for ASSM tablespace and needs more research to understand that. In test case #1 above, HWM for the table was set at file_id 6, block_id 80 for 640 blocks extent starting at file 6, block_id 9. That is, 70 blocks were below HWM initially. After allocating an extent with instance 1 keyword, HWM was increased to end of both extents. Meaning, HWM was bumped up by 1209 blocks.

Yong has a test case for this: http://yong321.freeshell.org/oranotes/Dbms_Space.txt

From segment header block, before allocating extent, HWM: 0x01800050 (dba for file 6,block 80)
alter table test_hw allocate extent (instance 1);

From segment header block, after allocating extent, HWM: 0x01802f89 (dba for file 6,block 12169)
Extent Map

0x0180000a length: 639

0x01802d09 length: 640

5. High HW enqueue contention is prevalent during Oracle applications upgrade also. During one of our recent upgrade from 11.5.8 to 11.5.0, there was heavy HW enqueue contention on sys.source\$ table. Only option was to increase `_bump_highwater_mark_count` during upgrade, to relieve excessive waits on HW contention. Of course, Oracle support must be contact for before adding any underscore parameters in production environment. Side effect of this is that, for smaller tables, full table scan might take longer since more blocks need to be scanned.