

Library cache locks and library cache pin waits:

I encountered few customer issues centered around library cache lock and library cache pin waits. Library cache lock and pin waits can hang instance, and in few cases, whole clusters of RAC instances can be hung due to library cache lock and pin waits.

Why Library cache locks are needed?

Library cache locks aka parse locks are needed to maintain dependency mechanism between objects and their dependent objects like SQL etc. For example, if an object definition need to be modified or if parse locks are to be broken, then dependent objects must be invalidated. This dependency is maintained using library cache locks. For example, if a column is dropped from a table then all SQLs dependent upon the table must be invalidated and reparsed during next access to that object. Library cache locks are designed to implement this tracking mechanism.

In a regular enqueue locking scenarios there is a resource (example TM table level lock) and sessions enqueue to lock that resource. Similarly, library cache locks uses object handles as resource structures and locks are taken on that resource. If the resources are not available in a compatible mode, then sessions must wait for library cache objects to be available.

Why Library cache pins are needed?

Library cache pins deals with current execution of dependent objects. For example, an underlying objects should not be modified when a session is executing or accessing a dependent object (SQL). So, before parse locks on a library cache object can be broken, library cache pins must be acquired in Exclusive mode and then only library cache objects can be dropped. If a session is executing an SQL, then library cache pins will not be available and there will be waits for library cache pins. Typically, this happens for long running SQL statement.

x\$kgllk, x\$kglpn and x\$kglob

Library cache locks and pins are externalized in three x\$ tables. x\$kgllk is externalizing all locking structures on an object. Entries in x\$kglob acts as a resource structure. x\$kglpn is externalizing all library cache pins.

x\$kglob.kglhdadr acts as a pointer to the resource structure. Presumably, kglhdadr stands KGL handle address. x\$kgllk acts as a lock structure and x\$kgllk.kglkhd points to x\$kglob.kglhdadr. Also, x\$kglpn acts as a pin structure and x\$kglpn.kglpnhdl points to x\$kglob.kglhdadr to pin a resource. To give an analogy between object locking scenarios, x\$kglob acts as resource structure and x\$kgllk acts as lock structures for library cache locks. For library cache pins, x\$kglpn acts as pin structure. x\$kglpn also pins that resource using kglpnhdl. This might be clear after reviewing the example below.

Test case

We will create a simple test case to create library cache locks and pin waits

```
create or replace procedure backup.test_kgllk
    (l_sleep in boolean , l_compile in boolean)
as
begin
    if (l_sleep ) then
        sys.dbms_lock.sleep(60);
    elsif (l_compile ) then
        execute immediate 'alter procedure test_kgllk compile';
    end if;
end;
/
```

In this test case above, we create a procedure and it accepts two boolean parameters: sleep and compile. Passing true to first argument will enable the procedure to sleep for a minute and passing true for the second argument will enable the procedure to recompile itself.

Let's create two sessions in the database and then execute them as below.

Session #1: exec test_kgllk (true, false); — Sleep for 1 minutes and no compile

Session #2: exec test_kgllk (false, true); — No sleep, but compile..

At this point both sessions are waiting. Following SQL can be used to print session wait details.

```
select
    distinct
        ses.ksusenum sid, ses.ksuseser serial#, ses.ksuudlna
username, ses.ksuseunm machine,
        ob.kglnaown obj_owner, ob.kglnaobj obj_name
        , pn.kglpncnt pin_cnt, pn.kglpnmod pin_mode, pn.kglpnreq pin_req
        , w.state, w.event, w.wait_Time, w.seconds_in_Wait
        -- lk.kglnaobj, lk.user_name, lk.kgllksnm,
        -- ,lk.kgllkhd1, lk.kglhdpar
        -- ,trim(lk.kgllkcnt) lock_cnt, lk.kgllkmod lock_mode, lk.kgllkreq
lock_req,
        -- ,lk.kgllkpns, lk.kgllkpnc, pn.kglpnhdl
from
    x$kglpn pn, x$kglob ob, x$ksuse ses
    , v$session_wait w
where pn.kglpnhdl in
(select kglpnhdl from x$kglpn where kglpnreq >0 )
and ob.kglhdadr = pn.kglpnhdl
and pn.kglpnuse = ses.addr
and w.sid = ses.indx
order by seconds_in_wait desc
/
```

Output of above SQL is:

pin	SID	SERIAL#	USERNAME	MACHINE	wait seconds	OBJ_OWNER	OBJ_NAME	cnt	mode	req	STATE	pin	pin
EVENT	time	in_wait											
268	12409	SYS		orap	SYS	TEST_KGLLK	3	2	0		WAITING		
PL/SQL lock timer	0			7									
313	45572	SYS		orap	SYS	TEST_KGLLK	0	0	3		WAITING		
library cache pin	0			3									
313	45572	SYS		orap	SYS	TEST_KGLLK	3	2	0		WAITING		
library cache pin	0			3									

1. Session 268 (session #1) is sleeping while holding library cache pin on test_kglk object (waiting on PL/SQL lock timer more accurately).
2. Session 313 is holding library cache pin in mode 2 and waiting for library cache pin in mode 3.

Obviously, session 313 is waiting for session 268 to release library cache pins. Since session 268 is executing, session 313 should not be allowed to modify test_kglk library cache object. That's exactly why library cache pins are needed.

Adding another session to this mix..

Let's add one more session to this mix..

```
exec test_kglk (false, true);
```

Output of above query is:

pin	pin	pin	SID	SERIAL#	USERNAME	MACHINE	wait seconds	OBJ_OWNER	OBJ_NAME	cnt
mode	req	STATE	EVENT	time	in_wait					
268	12409	SYS		oraperf	SYS	TEST_KGLLK	3			
2	0	WAITING	PL/SQL lock timer	0		34				
313	45572	SYS		oraperf	SYS	TEST_KGLLK	0			
0	3	WAITING	library cache pin	0		29				
313	45572	SYS		oraperf	SYS	TEST_KGLLK	3			
2	0	WAITING	library cache pin	0		29				
442	4142	SYS		oraperf	SYS	TEST_KGLLK	0			
0	2	WAITING	library cache pin	0		3				

Well, no surprise there. New session 442 also waiting for library cache pin. But, notice the request mode for session 442. It is 2. Session 442 needs that library cache pin in share mode to start execution. But 313 has already requested that library cache pin in mode 3. A queue is building up here. Many processes can queue behind session 313 at this point leading to an hung instance.

library cache locks..

Let's execute same package but both with same parameters.

```
Session #1: exec test_kgllk(false, true);
Session #2: exec test_kgllk(false, true);
```

Rerunning above query tells us that session 313 is waiting for the self. Eventually, this will lead library cache pin self deadlock.

Library cache pin holders/waiters

```
-----
```

pin						wait seconds		pin
SID	SERIAL#	USERNAME	MACHINE	OBJ_OWNER	OBJ_NAME	time	in_wait	cnt
mode	req	STATE	EVENT					

313	45572	SYS	oraperf	SYS	TEST_KGLLK			0
0	3	WAITING	library cache pin	0	26			
313	45572	SYS	oraperf	SYS	TEST_KGLLK			3
2	0	WAITING	library cache pin	0	26			

```
-----
```

Wait, what happened to session #2? It is not visible in x\$kgllpn. Querying v\$session_wait shows that Session #2 is waiting for library cache lock. We will run yet another query against x\$kgllk to see library cache lock waits. Querying x\$kgllk with the query below:

```
select
distinct
  ses.ksusenum sid, ses.ksuseser serial#, ses.ksuudlna
username,KSUSEMNM module,
  ob.kglnaown obj_owner, ob.kglnaobj obj_name
  ,lk.kgllkcnt lck_cnt, lk.kgllkmod lock_mode, lk.kgllkreq lock_req
  , w.state, w.event, w.wait_Time, w.seconds_in_Wait
from
  x$kgllk lk, x$kglob ob,x$ksuse ses
  , v$session_wait w
where lk.kgllkhd1 in
(select kgllkhd1 from x$kgllk where kgllkreq >0 )
and ob.kglhdadr = lk.kgllkhd1
and lk.kgllkuse = ses.addr
and w.sid = ses.indx
order by seconds_in_wait desc
/
```

Library cache lock holders/waiters

```

-----
lock
      SID SERIAL# USERNAME      MODULE      OBJ_OWNER OBJ_NAME      LCK_CNT mode
req STATE      EVENT
-----
313    45572 SYS          wsqfincla  SYS        TEST_KGLLK      1      1
0 WAITING  library cache pin          0          29
313    45572 SYS          wsqfincla  SYS        TEST_KGLLK      1      3
0 WAITING  library cache pin          0          29
268    12409 SYS          wsqfincla  SYS        TEST_KGLLK      0      0
2 WAITING  library cache lock          0          12
268    12409 SYS          wsqfincla  SYS        TEST_KGLLK      1      1
0 WAITING  library cache lock          0          12
-----

```

Session 313 is holding library cache lock on that object in mode 3 and session 268 is requesting lock on that library cache object in mode 2. So, session 268 is waiting for library cache lock while session 313 is waiting for library cache pin (self). Again, point here is that session 268 is trying to access library cache object and need to acquire library cache lock in correct mode. That library cache lock is not available leading to a wait.

Complete script can be downloaded from my [script archive](#).

RAC, library cache locks and pins

Things are different in RAC. Library cache locks and pins are global resources controlled by GES layer. So, these scripts might not work if these library cache lock and pin waits are global events. Let's look at what happens in a RAC environment

```

exec test_kgllk ( false, true); -- node 1
exec test_kgllk ( false, true); -- node 2

```

In node1, only one session is visible.

Library cache pin holders/waiters

```

-----
pin pin
      SID SERIAL# USERNAME      MACHINE      OBJ_OWNER OBJ_NAME      cnt
mode req STATE      EVENT
-----
268    12409 SYS          oraperf     SYS        TEST_KGLLK      0
0      3 WAITING  library cache pin          0          18
268    12409 SYS          oraperf     SYS        TEST_KGLLK      3
2      0 WAITING  library cache pin          0          18
In node 2, only requestor of the lock is visible.
-----

```

```

-----
lock
      SID SERIAL# USERNAME      MODULE      OBJ_OWNER OBJ_NAME      LCK_CNT mode
req STATE      EVENT
-----
377    43558 SYS          wsqfinc2a  SYS        TEST_KGLLK      0      0
2 WAITING  library cache lock          0          86
-----

```

Essentially, this script does not work in a RAC environment since it accesses x\$ tables directly, which are local to an instance. To understand the issue in a RAC environment we need to access gv\$ views, based on x\$kgllk, x\$kglpn etc. But, I don't see gv\$ views over these x\$ tables.

Nevertheless, we can see lockers and waiters accessing gv\$ges_blocking_enqueue to understand locking in RAC.

```

1 select inst_id, handle, grant_level, request_level, resource_name1, resource_name2,
pid , transaction_id0, transaction_id1
2* ,owner_node, blocked, blocker, state from gv$ges_blocking_enqueue
SQL> /

```

INST_ID	HANDLE	GRANT_LEV	REQUEST_L	RESOURCE_NAME1	RESOURCE_NAME2	PID
2	00000008DD779258	KJUSERNL	KJUSERPR	[0x45993b44][0x3a1b9eee],[LB]	1167670084,974888686,LB	8700
		0	1			
						OPENING
1	00000008E8123878	KJUSEREX	KJUSEREX	[0x45993b44][0x3a1b9eee],[LB]	1167670084,974888686,LB	12741
		0	0			
						GRANTED

We can see that PID 12741 from instance 1 is holding a library cache global lock [LB]. Global resource in this case is [0x45993b44][0x3a1b9eee],[LB] which uniquely identifies a library cache object at the cluster level. Grant_level is KJUSEREX or Exclusive level and request_level from node 2 is KJUSERPR which is Protected Read level. PID 8700 in node 2 is waiting for library cache lock held by PID 12741 in node1. Using this output and our script output, we can pin point which process is holding library cache lock or pin. While Library cache locks are globalized as global locks in the range of [LA] - [LZ], Library cache pins are also globalized as lock types in the range [NA]-[NZ].