

High global cache waits on tab\$?

Yes, you read it correct. That shocked me too.

I was trying to understand global cache waits for a client. Naturally, I queried statspack tables and analyzed the data for just one day using a script. Surprisingly, tab\$ came as top consumer of global cache waits. I was shocked by the revelations and couldn't believe it! If something doesn't make sense, look more closely, right?

Global cache waits

Database version is 9i. Statspack table stats\$seg_stat is the source for this script and that table is populated from v\$segment_stats. So, these column values (global_cache_cu_blocks_served and global_cache_cr_blocks_served) are cumulative. To find statistics for a specific period, we need to subtract column value of prior row from current row. Analytic function lag can be useful for this.

In this analytic function printed below, partitioning clause uses instance_number, startup_time and dataobj#. All rows with same value for these three columns will be considered in one data partition and then rows ordered by snap_id within that data partition. Lag will pull the row from prior snap_id in that partition. Then, we subtract from current value to get the difference. Please refer to this paper: [Performance tuning with SQL new features - paper](#) for more information about analytic functions.

```
...
    global_cache_cu_blocks_served -
    lag(global_cache_cu_blocks_served,1,0) over (partition by
instance_number,startup_time, dataobj#, obj#
            order by  snap_id ) global_cache_cu_blocks_served,
...

```

Script and output

Complete script printed below and running that against a database.

```
with stats1 as(
  select /*+ materialize */ snap_time,obj#,dataobj#, dbid,
global_cache_cu_blocks_served
  from (
    select
      snap_time , dataobj#,obj#,snap_id, dbid, instance_number,
startup_time,
      global_cache_cu_blocks_served -
      lag(global_cache_cu_blocks_served,1,0) over
        (partition by instance_number,startup_time, dataobj#
          order by  snap_id ) global_cache_cu_blocks_served
    from (

```

```

select /*+ leading (snap) parallel (snap 5) parallel ( e 5) */
snap.snap_time , e.dataobj#, e.obj#,
e.global_cache_cu_blocks_served, e.snap_id, e.dbid,
e.instance_number, snap.startup_time
from
perfstat.stats$seg_stat e, perfstat.stats$snapshot snap
where
e.snap_id =snap.snap_id
and e.instance_number = snap.instance_number
and e.dbid = snap.dbid
and trunc(snap.snap_time) = trunc(sysdate-1) -- one day ago
order by instance_number, startup_time, dataobj#, snap_id
)
)
select * from
( select trunc( snap_time) , dataobj#, obj#,
sum(GLOBAL_CACHE_CU_BLOCKS_SERVED)
from stats1 s
group by trunc( snap_time) , dataobj#, obj#
order by 4 desc
) where rownum <21
/

```

TRUNC(SNAP_TIME)	DATAOBJ#	OBJ#	SUM(GLOBAL_CACHE_CU_BLOCKS_SERVED)
24-FEB-09	2	4	1607634321
24-FEB-09	5734032	41995	342437603
24-FEB-09	1297689	44933	322657052
24-FEB-09	6081089	34450	320872029
....			

Interesting to note that dataobj# is 2 and obj# is 4 which is associated with tab\$ table. I couldn't believe this. I can not imagine tab\$ being a top consumer of global cache current blocks. It is possible, but I doubt it.

```

1* select obj#, dataobj#, owner# , name from sys.obj$ where
dataobj#=2 and obj#=4
SQL> /

```

OBJ#	DATAOBJ#	OWNER#	NAME
4	2	0	TAB\$

So, I decided to break the SQL in to smaller piece and looked at data before aggregation. There it is!

I was under the impression that dataobj# (same as dba_objects.data_object_id) is unique. I knew that data_object_id is updated after every truncate of that table, but never realized that many objects in sys schema has dataobj# set to 2. Yikes!

```

1* select obj#, dataobj#, owner# , name from sys.obj$ where
dataobj#=2
SQL> /

```

OBJ#	DATAOBJ#	OWNER#	NAME
20	2	0	ICOL\$
19	2	0	IND\$
5	2	0	CLU\$
2	2	0	C_OBJ#
21	2	0	COL\$
4	2	0	TAB\$
241	2	0	COLTYPE\$
244	2	0	ATTRCOL\$

```

...
17 rows selected.

```

This misconception affected SQL functionality. Essentially, partitioning clause in lag is incorrect. Partitioning clause in the lag statement does not include obj# causing this issue. Adding obj# to the lag partitioning clause. [Reprinting lag clause with obj# below.]

```

...
global_cache_cu_blocks_served -
lag(global_cache_cu_blocks_served,1,0) over (partition by
instance_number,startup_time, dataobj#, obj#
order by snap_id ) global_cache_cu_blocks_served
...

```

Finally, I modified the SQL to join with dba_objects to print object_names. Complete script can be found here: [sp_gc_obj.sql](#).

Summary

After fixing the bug with partitioning clause in lag statement, now SQL returns correct object_name.

OWNER	OBJECT_NAME	GC_CU_BLOCKS_SERVED
ONT	OE_ORDER_LINES_ALL	342437603
PO	PO_REQUISITION_LINES_ALL	322657052
APPLSYS	FND_CONCURRENT_REQUESTS	320872029

```

...
20 rows selected.

```