vahidusefzadeh@gmail.com

RAC -Dynamic Remastering

در محیط RAC هر instance بخشی از GRD را در SGA خود جای می دهد و همانطور که می دانید GRD هم حاوی اطلاعاتی از قبیل data block address، SCN، past image، current image و ... می باشد پس در نتیجه هر instance، اطلاعات مجموعه ای از بلاکها را کنترل می کند و master آنها می باشد و متقابلا هر بلاک هم یک نود مستر دارد به طوری که اگر مستر فعلی بلاک دچار مشکل شود، نود دیگری مسئولیت این بلاکها را به عنوان مستر بر عهده می گیرد.

معمولا بلاکها به صورت گروهی و پیوسته به یک نود تخصیص داده می شوند به طوری که در حالت پیش فرض، هر ۱۲۸ بلاک ازآن یک نود می شود که با پارامتر مخفی _lm_contiguous_res_count می توان این موضوع را مدیریت نمود. مقدار جاری این پارامتر، با پرس و جوی زیر مشخص می شود:

```
select * from (Select Ksppinm, Ksppstvl, Ksppdesc FROM X$Ksppi X, X$Ksppcv Y WHERE X.Indx =
Y.Indx AND TRANSLATE(ksppinm,'_','#') like '#%' )u where ksppinm like '%_lm_contiguous%';
```

حال فرض کنید instanceای قصد دسترسی و اصلاح بلاکی را دارد که مستر آن بلاک نیست در این صورت باید با استفاده از GCS به نود مستر آن بلاک پیامی را برساند و در نهایت مجوز لازمه را از مستر دریافت کند. اگر این اتفاق به دفعات رخ دهد، ممکن است waitهایی به مثل gc cr grant 2-way و gc current grant 2-way به کررات در سیستم دیده شوند که این waitها با تغییر مستر بلاک به حداقل خواهند رسید و به طور مثال زمانی که بلاک در بافرکش هیچ instanceای نباشد، instance به دلیل مستر بودن، زمانی را برای گرفتن grant صرف نمی کند و عمده انتظارش بخاطر انتقال بلاک از دیسک به بافرکش خواهد بود.

- ✓ **یک نمونه:**

```
SQL> oradebug setmypid
```

vahidusefzadeh@gmail.com

SQL> alter session set tracefile_identifier='useftrace';

SQL> oradebug tracefile_name

/u02/oracle/diag/rdbms/rac/rac2/trace/rac2_ora_10633_useftrace.trc

SQL> delete from usef.DRM1 where rownum=1;

با رجوع به فایل trace، خواهیم دید که waitای در این زمینه رخ داده چون نود دیگری مستر این جدول بوده است:

WAIT #140143856009776: nam='gc cr grant 2-way' ela= 1676 p1=1 p2=506 p3=1 obj#=57 tim=1456907912697406

در صورتی که اگر همین دستور در نود مستر این جدول اجرا شود، این wait دیده نخواهد شد و مدل waitها متفاوت خواهد بود:

SQL> delete from usef.DRM1 where rownum=1;

WAIT #140428227776632: nam='gc cr disk read' ela= 2266 p1=4 p2=450 p3=4 obj#=91552 tim=1456902704915875

WAIT #140428227776632: nam='db file sequential read' ela= 19599 file#=4 block#=450 blocks=1 obj#=91552 tim=1456902704937020

WAIT #140428227776632: nam='gc cr disk read' ela= 1436 p1=4 p2=466 p3=4 obj#=91554 tim=1456902704955443

WAIT #140428227776632: nam='db file sequential read' ela= 34254 file#=4 block#=466 blocks=1 obj#=91554 tim=1456902704991502

از اوراکل 10gR1 ویژگی جدیدی ارائه شد که با استفاده از آن ممکن بود مستر یک بلاک به طور خودکار تغییر کند و هر نودی که در بازه زمانی مشخص و به اندازه‌ای مشخص بیشتر از بقیه نودها به بلاکی از طریق gc دسترسی پیدا می کرد، کل فایل مربوط به آن بلاک، به این نود تخصیص داده می شد این ویژگی **DRM**(Dynamic REMASTERING) نام دارد.

vahidusefzadeh@gmail.com

این ویژگی در نسخه 10gR1 بعضا با باگهایی همراه بود که مسئولین بانک را مجبور می‌کرد تا این ویژگی را غیر فعال کنند به طور مثال ممکن بود با ایجاد خطای زیر سبب down شدن بانک شود:

ORA-00481: LMON process terminated with error

در نسخه 10gR2، بسیاری از مشکلات مربوط به این ویژگی مرتفع شد و به جای اینکه REMASTERING در سطح فایل انجام شود، به سطح object تقلیل پیدا کرد.

برای تنظیم کیفیت کار DRM، مجبور هستیم تا با پارامترهای مخفی اوراکل کار کنیم که اطلاعات مستند شده بسیار کمی در مورد آنها وجود دارد از نمونه برای غیرفعال کردن این ویژگی در نسخه 10g، باید پارامترهای زیر را در spfile تنظیم کنیم:

_gc_affinity_time : فاصله زمانی که لازم است تا بررسی شود که آیا REMASTERING لازم است یا خیر.

_gc_undo_affinity: DRM در دو سطح external و internal انجام می شود که منظور از external همان بلاکهای حاوی اطلاعات کاربران می باشد و منظور از internal داده های undoها می باشند که می توانند REMASTERING شوند. این پارامتر به سطح داخلی REMASTERING اشاره دارد.

_gc_affinity_time=0

_gc_undo_affinity=FALSE

همچنین در محیط 10g به دلیل فعال بودن DRM، ممکن است با دو wait با نام های gc master و gcs drm service freeze مواجه شویم(به دلیل GRD freeze)، که می توانیم با بهینه کردن مقدار پارامتر _gc_affinity_limit، این waitها را کم رنگ کنیم.

زمانی که REMASTERING انجام می شود، چند پروسس در این زمینه نقش اصلی را ایفا می کنند:

vahidusefzadeh@gmail.com

LCK0: این پروسس اطلاعات و آمارهای سطح شی را مورد بررسی قرار می دهد(تعداد رجوع به object از طریق نودهای مختلف) و بررسی می کند چه زمانی باید REMASTERING انجام شود.

LMD0: بعد از شناسایی شی برای REMASTERING، درخواست در صف قرار می گیرد و پروسس LMD0 این درخواست را می خواند. pkey به object_id شی ای اشاره دارد که باید REMASTERING شود.

```
Begin DRM(13) (swin 0) - AFFINITY transfer pkey 91515.0 to 2 oscan 1.1
kjiobjscn 1
ftd (30) received from node 1 (12 0.0/0.0)
all ftds received
```

LMON و LMS: نهایتا این این دو پروسس عمل REMASTERING را انجام می دهند. پروسس LMON عمل GRD freeze را انجام می دهد که بعد از آن، BL lock ممکن نخواهد بود این عمل برای sync کردن instanceهای مختلف با شرایط جدید صورت می گیرد و نهایتا هم پروسس مذکور GRD را از freeze خارج می کند.drm freeze در سیستم های با حجم کاری بالا، ممکن است با ایجاد waitهایی طولانی از قبیل gcs drm freeze enter server event، بر روی کارایی بانک اثر منفی بگذارد.

```
* DRM RCFG called (swin 0)
 CGS recovery timeout = 85 sec
Begin DRM(12) (swin 0)
 object id 91515.0, objscan 1.1, move affinity from ١ to ٢
 Total pkey count in this drm  1
* drm quiesce
2016-02-29 14:43:56.318468 : DRM(12) resources quiesced [0-1023], rescount 1880
2016-02-29 14:43:56.319058 : DRM(12) local converts quiesced [0-1023], lockcount 0, bucket 0
* drm sync 1
* drm freeze
* DRM(12) window 1, drm freeze complete.
```

vahidusefzadeh@gmail.com

LMS:

DRM(13) quiesced basts [0-1023]

* lms 1 finished parallel drm freeze in DRM(1۲) window 1, pcount 22

DRM(1۲) win(1) lms 1 finished drm freeze

نکته: از دیگر اصطلاحاتی که باید در این زمینه به آن توجه نمود، اصطلاح owner بلاک می باشد این اصطلاح به نودی که بلاک در بافرکش آن موجود است، اشاره دارد و حسابش از مستر جدا می‌باشد. برای تعیین owner و master بلاکها می توانیم از پرس و جوی زیر کمک بگیریم:

```
select kj.*, le.le_Addr from(select kjblname, kjblname2, kjblowner, kjblmaster, kjbllockp,substr (
kjblname2,  instr(kjblname2,',')+1,   instr(kjblname2,',',1,2)-instr(kjblname2,',',1,1)-1)/65536 fl, substr (
kjblname2, 1, instr(kjblname2,',')-1) blk from x$kjbl) kj, x$le le where le.le_kjbl = kj.kjbllockp order by
le.le_addr ;
```

کشف مستر نود بلاکهای یک جدول:

همانطور که در ابتدا گفته شد، در آغاز کار MASTERING در سطح بلاک انجام می شود پس ممکن است بلاکهای یک جدول چندین مستر داشته باشند پرس و جویی که در ادامه خواهیم دید، مسترنود بلاکهای یک جدول را به ما نشان خواهد داد البته به شرطی که این جدول حداقل یکبار توسط آن نود مورد دسترسی قرار گرفته باشد به همین دلیل ابتدا با دستور زیر، به جدول دسترسی پیدا می کنیم:

```
select * from u where a<10;
```

سپس با پرس و جوی زیر، مسترِ بلاکهای این جدول را خواهیم یافت:

```
With OBJ_IDS As (Select DATA_Object_Id OBJECT_ID From Dba_Objects  Where Object_Name = 'U'),
Addr As (Select /*+materialize */
Le_Addr, class, state From X$Bh, OBJ_IDS Where Object_Id = Obj),
Hexnames As (Select Rtrim(B.Kjblname, ' '||chr(0)) Hexname From X$Le A, X$Kjbl B, Addr Where
A.Le_Kjbl=B.Kjbllockp and class = 1 and state <> 3 And A.Le_Addr = Addr.Le_Addr)
Select A.Master_Node Mast, Count(*) From Gv$Dlm_Ress  A, Hexnames H Where
Rtrim(A.Resource_Name, ' '||chr(0)) = H.Hexname Group by A.Master_Node ;
```

vahidusefzadeh@gmail.com

در ادامه سه حالت ممکن برای REMASTERING را مورد بررسی قرار خواهیم داد.

REMASTERING به صورت دستی:

در صورتی که بخواهیم بصورت دستی مستر یک شی را تغییر دهیم، می توانیم مراحل زیر را طی کنیم:

۱. ابتدا جدولی با اسم usef1 ساختیم که object_id آن برابر است با 91535:

```
create table usef1 as select * from v$datafile;

 SQL> select object_id,owner,object_type from dba_objects where object_name='USEF1' and
object_type='TABLE';

 OBJECT_ID              OWNER                  OBJECT_TYPE
----------          --------------------------          -------------------
   91535                SYS                              TABLE
```

۲. با دستور زیر، تنها یک نود، مستر کل بلاکهای جدول usef1 خواهد شد

```
SQL> oradebug setmypid

Statement processed.

SQL> oradebug lkdebug -m pkey 91535

Statement processed.
```

توضیح اینکه ابزار oradebug بعنوان یک ابزار تحلیلی، علاوه بر توانایی کمک برای رفع خطا در سیستم، می تواند برای کارایی بیشتر سیستم هم مورد استفاده قرار بگیرد. در دستور بالا، از پارامتر lkdebug این دستور استفاده شده تا بتوانیم GES(global enqueue service) را صدا بزنیم برای دیدن همه سوییچهای مربوط به این پارامتر، می توانیم از دستور oradebug lkdebug help کمک بگیریم سوییچی که در دستور بالا استفاده شد، - m pkey بود که برای تغییر نود جاری به عنوان master یک شی به کار می رود.

پرس و جوی زیر نشان می دهد که مستر این جدول از کدام نود به کدام نود تغییر کرده است:

```
SQL> select * from V$GCSPFMASTER_INFO where data_object_id=91535;
```

vahidusefzadeh@gmail.com

FILE_ID	DATA_ OBJECT_ID	GC_MASTERIN	CURRENT_MASTER	PREVIOUS_MASTER	REMASTER_CNT
0	91353	Affinity	1	0	2

همانطور که می بینید، خروجی دستور بالا نشان می دهد که مستر جدول usef1، قبلا نود اول بود و فعلا به نود دوم تغییر کرده است تعداد دفعات REMASTERING برای این جدول، دو بار بوده است.

REMASTERING به صورت اتوماتیک:

در ادامه سناریویی آورده شده که منجر به تغییر مستر جدول به صورت اتوماتیک خواهد شد:

۱. ابتدا جدولی با اسم DRM1 می سازیم و اطلاعاتی را در آن درج می کنیم.

```
SQL> create table usef.DRM1 as select * from aud$;

declare

begin

for i in 1..3 loop

insert into usef.DRM1 select * from usef.DRM1;

end loop;

end;
```

۲. برای ادامه کار لازم است تا object_id این جدول را داشته باشیم:

```
SQL> select owner, data_object_id, object_name, object_id from dba_objects where owner = 'USEF'
and object_name = 'DRM1';

OWNER DATA_OBJECT_ID OBJECT_NAME      OBJECT_ID

----- -------------- --------------- ----------

USEF         91550 DRM1              91550
```

vahidusefzadeh@gmail.com

۳. کیفیت انجام DRM معمولا با پارامترهای مخفی کنترل می شود و همانطور که می دانید این پارامترها مستندات اوراکلی بسیار اندکی دارند و با "_" از پارامترهای غیرمخفی جدا می شوند. در این زمینه دو پارامتر مهم وجود دارد که به این اسم می باشند(در اوراکل 11g):

_gc_policy_time: این پارامتر مشخص می کند که چند دقیقه یکبار باید آمارها بررسی شوند تا اگر برای شیِ خاصی لازم بود، REMASTERING انجام شود. مقدار پیش فرض آن ده دقیقه می باشد.

_gc_policy_minimum: حداقل میزان رخ دادن gc activity توسط یک instance غیر مستر، به چه اندازه باشد تا آن شی صلاحیت REMASTERING را داشته باشد. مقدار پیش فرض آن برابر با 2500 می باشد.

برای دیدن مقدار جاری این دو پارامتر، می توانیم از پرس و جوی زیر استفاده کنیم:

```
select * from (Select Ksppinm, Ksppstvl, Ksppdesc FROM X$Ksppi X, X$Ksppcv Y WHERE X.Indx = Y.Indx
AND TRANSLATE(ksppinm,'_','#') like '#%' )u where ksppinm like '%_gc_policy%';
```

برای اینکه در ادامه این سناریو، REMASTERING خودکار را شاهد باشیم، با دو دستور زیر دو پارامتر مذکور را به گونه ای تنظیم می کنیم که در حداقل زمان ممکن، REMASTERING برای objectها رخ دهد:

```
SQL> alter system set "_gc_policy_minimum" = 10 scope=spfile;

System altered.

SQL> alter system set "_gc_policy_time" = 1 scope=spfile;

System altered.
```

برای اعمال این تغییرات، باید بانک مجددا استارت شود.

```
[oracle@rac1 ~]$ srvctl stop database -d rac

[oracle@rac1 ~]$ srvctl start database -d rac
```

۴. با اجرای دستورات زیر قصد داریم نود اول، مستر همه بلاکهای جدول DRM1 شود پس دستور زیر را در نود اول اجرا می کنیم:

vahidusefzadeh@gmail.com

SQL> oradebug setmypid

Statement processed.

SQL> oradebug lkdebug -m pkey 91550

Statement processed.

همانطور که پرس و جوی زیر نشان می دهد، جدول DRM1 سه بار REMASTERING شده که مستر فعلی آن، نود اول می باشد:

```
SQL> select o.object_name, m.CURRENT_MASTER,m.PREVIOUS_MASTER, m.REMASTER_CNT from
dba_objects o, v$gcspfmaster_info m where o.data_object_id=91550   and m.data_object_id = 91550   ;

OBJECT_NAM CURRENT_MASTER PREVIOUS_MASTER REMASTER_CNT

----------          --------------          ---------------       ------------

DRM1                    0                              1             3
```

۵. در این مرحله قصد داریم تا دسترسی به جدول DRM1 را از طریق نود دوم به حدی بالا ببریم تا REMASTERING به صورت خودکار انجام شود:

```
declare

begin

for i in 1..3 loop

insert into usef.DRM1 select * from usef.DRM1;

end loop;

end;
```

در حین اجرای دستور بالا، دستور زیر را در session دیگری از این نود دوم اجرا می کنیم که نشان می دهد تعداد دفعات lock در سطح object چند بار بوده است که XOPENS بیانگر exclusive BL lock و SOPENS هم بیانگر shared BL lock می باشند:

SQL> select inst_id, sopens, xopens from x$object_policy_statistics where object= 91550;

```
  INST_ID    SOPENS    XOPENS

---------- ---------- ----------
```

vahidusefzadeh@gmail.com

```
         2          0       1024
```

SQL> /

```
   INST_ID     SOPENS     XOPENS
---------- ---------- ----------
         2          0       1517
```

همانطور که می بینید، نهایتا DRM1 از نود اول به نود دوم، REMASTER شده است:

SQL> select o.object_name, m.CURRENT_MASTER,m.PREVIOUS_MASTER, m.REMASTER_CNT from dba_objects o, v$gcspfmaster_info m where o.data_object_id=91550 and m.data_object_id = 91550;

```
OBJECT_NAM CURRENT_MASTER PREVIOUS_MASTER REMASTER_CNT
---------- -------------- --------------- ------------
DRM1                    1               0            4
```

REMASTERING در اثر crash:

حالتی را فرض کنید که نود دوم مسترِ جدول DRM1 است و به هر دلیلی down می شود:

SQL> shutdown abort

```
ORACLE instance shut down.
```

در این صورت، به صورت خودکار REMASTERING انجام می شود و نود اول نقش مستر آن جدول را بازی خواهد کرد:

SQL> select o.object_name, m.CURRENT_MASTER,m.PREVIOUS_MASTER, m.REMASTER_CNT from dba_objects o, v$gcspfmaster_info m where o.data_object_id=91550 and m.data_object_id = 91550;

```
OBJECT_NAM CURRENT_MASTER PREVIOUS_MASTER REMASTER_CNT
---------- -------------- --------------- ------------
DRM1                    0               1            5
```

vahidusefzadeh@gmail.com

گزارش AWR در مورد REMASTERING:

Dynamic Remastering Stats

- times are in seconds
- Affinity objects - objects mastered due to affinity at begin/end snap

Name	Total	per Remaster Op	Begin Snap	End Snap
remaster ops	4	1.00		
remastered objects	4	1.00		
replayed locks received	0	0.00		
replayed locks sent	266	66.50		
resources cleaned	0	0.00		
remaster time (s)	1.3	0.33		
quiesce time (s)	0.2	0.06		
freeze time (s)	0.1	0.04		
cleanup time (s)	0.1	0.03		
replay time (s)	0.2	0.04		
fixwrite time (s)	0.1	0.03		
sync time (s)	0.5	0.13		
affinity objects			1	1