【故障处理】序列 cache 值过小导致 CPU 利用率过高

1.1 BLOG 文档结构图

1.2 前言部分

1.2.1 导读和注意事项

各位技术爱好者，看完本文后，你可以掌握如下的技能，也可以学到一些其它你所不知道的知识，~O(∩_∩)O~：

① enq: SQ - contention 等待事件的解决

② 一般等待事件的解决办法

③ DFS lock handle 等待事件

④ 与序列有关的等待事件

Tips：

① 本文在 ITpub（http://blog.itpub.net/26736162）、博客园（http://www.cnblogs.com/lhrbest）和微信公众号（xiaomaimiaolhr）有同步更新

② 文章中用到的所有代码，相关软件，相关资料请前往小麦苗的云盘下载

（http://blog.itpub.net/26736162/viewspace-1624453/）

③ 若文章代码格式有错乱，推荐使用搜狗、360 或 QQ 浏览器，也可以下载 pdf 格式的文档来查看，pdf 文档下载地址：http://blog.itpub.net/26736162/viewspace-1624453/，另外 itpub 格式显示有问题，可以去博客园地址阅读

④ 本篇 BLOG 中命令的输出部分需要特别关注的地方我都用灰色背景和粉红色字体来表示，比如下边的例子中，thread 1 的最大归档日志号为 33，thread 2 的最大归档日志号为 43 是需要特别关注的地方；而命令一般使用黄色背景和红色字体标注；对代码或代码输出部分的注释一般采用蓝色字体表示。

```
  List of Archived Logs in backup set 11
  Thrd Seq     Low SCN    Low Time            Next SCN   Next Time
  ---- ------- ---------- ------------------- ---------- ---------
  1    32      1621589    2015-05-29 11:09:52 1625242    2015-05-29 11:15:48
  1    33      1625242    2015-05-29 11:15:48 1625293    2015-05-29 11:15:58
  2    42      1613951    2015-05-29 10:41:18 1625245    2015-05-29 11:15:49
  2    43      1625245    2015-05-29 11:15:49 1625253    2015-05-29 11:15:53

[ZHLHRDB1:root]:/>lsvg -o
T_XDESK_APP1_vg
rootvg
[ZHLHRDB1:root]:/>
00:27:22 SQL> alter tablespace idxtbs read write;

====》2097152*512/1024/1024/1024=1G
```

本文如有错误或不完善的地方请大家多多指正，ITPUB 留言或 QQ 皆可，您的批评指正是我写作的最大动力。

1.3 故障分析及解决过程

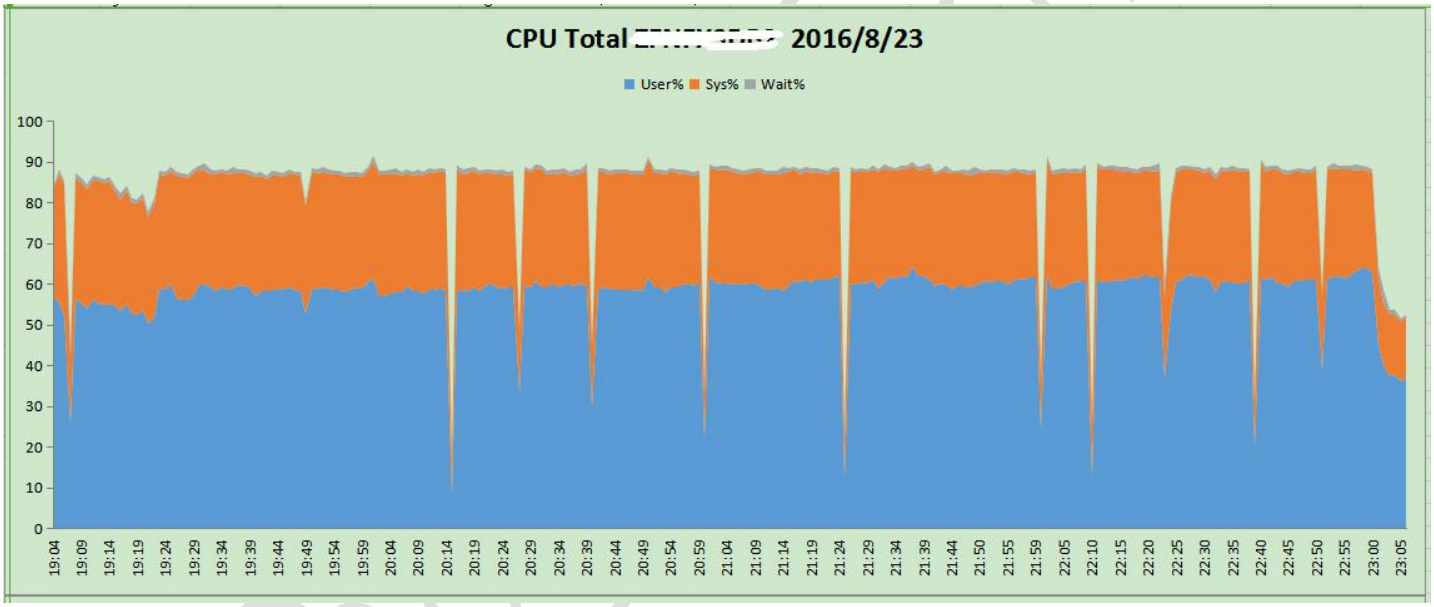
1.3.1 故障环境介绍

项目	source db

db 类型	RAC
db version	10.2.0.5.0
db 存储	ASM
OS 版本及 kernel 版本	AIX 64 位 6.1.0.0

1.3.2 故障发生现象及报错信息

早上同事过来跟我说昨天有一套数据库做测试的时候，CPU 利用率很高，他已经抓取了 CPU 和 AWR，让我帮忙分析分析，首先发生问题的时间段是 19 点到 23 点，nmon 数据截图如下：



可以看到 CPU 的利用率是非常高的，下边我们来看看 AWR 中的数据：

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
[illegible]	1654173250	[illegible]	1	10.2.0.5.0	YES	[illegible]

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	1418	23-Aug-16 19:04:41	515	3.4
End Snap:	1424	23-Aug-16 23:09:41	499	2.8
Elapsed:		245.00 (mins)		
DB Time:		6,722.27 (mins)		

Top 5 Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
DFS lock handle	1,640,153	80,725	49	20.0	Other
enq: SQ - contention	172,254	69,652	404	17.3	Configuration
CPU time		50,650		12.6	
gc buffer busy	3,134,968	30,426	10	7.5	Cluster
log file sync	11,550,206	22,607	2	5.6	Commit

其它的项目就不列出了，从等待事件中可以很明显的看出 enq: SQ - contention 和 DFS lock handle 这 2 个等待事件异常。Top 5 Timed Events 这个部分也是 AWR 报告中非常重要的部分，从这里可以看出等待时间在前五位的是什么事件，基本上就可以判断出性能瓶颈在什么地方。通常，在没有问题的数据库中，CPU time 总是列在第一个。在这里，**enq: SQ - contention 等待了 172254 次，等待时间为 69652 秒，平均等待时间为 69652/172254=404 毫秒，等待类别为 Configuration 即配置上的等待问题。**

1.3.3 故障分析

根据 AWR 报告的内容，我们知道只要解决了 enq: SQ - contention 和 DFS lock handle 这 2 个等待事件即可解决问题。那么我们首先来了解一些关于这 2 个等待事件的知识。

==========

enq: SQ - contention/row cache lock/DFS lock handle 这三个等待事件都与 Oracle 的 Sequence 有关。

```
SELECT *
  FROM V$EVENT_NAME
 WHERE NAME IN
       ('row cache lock', 'enq: SQ - contention', 'DFS lock handle');
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
200	2322460838	enq: SQ - contention	name|mode	object #	0	3290255840	2	Configuration
210	1714089451	row cache lock	cache id	mode	request	3875070507	4	Concurrency
360	3595075359	DFS lock handle	type|mode	id1	id2	1893977003	0	Other

使用如下的 SQL 我们可以查询到锁的名称和请求的 MODE，表的 mode 值参考表格：

```
select chr(bitand(p1,-16777216)/16777215)||
chr(bitand(p1, 16711680)/65535) "Lock",
bitand(p1, 65535) "Mode"
from v$session_wait
where event = 'DFS enqueue lock acquisition';
```

Table C-1 Lock Mode Values

Mode Value	Description
1	Null mode
2	Sub-Share
3	Sub-Exclusive
4	Share
5	Share/Sub-Exclusive
6	Exclusive

```
SELECT * FROM V$LOCK_TYPE D WHERE D.TYPE IN ('SV','SQ');
```

TYPE	NAME	ID1_TAG	ID2_TAG	IS_USER	DESCRIPTION
SQ	Sequence Cache	object #	0	NO	Lock to ensure that only one process can replenish the sequence cache
SV	Sequence Ordering	object #	0	NO	Lock to ensure ordered sequence allocation in RAC mode

Oracle 为了管理 Sequence 使用了以下三种锁。

① **row cache lock**:在调用 SEQUNECE.NEXTVAL 过程中，将数据字典信息进行物理修改时获取。赋予了 **NOCACHE** 属性的 SEQUENCE 上发生，等待事件为 **row cache lock。**

② **SQ 锁**:在内存上缓存（CACHE）的范围内，调用 SEQUENCE.NEXTVAL 期间拥有此锁。赋予了 CACHE 属性的 SEQUENCE 上发生。赋予了 CACHE 属性的 SEQUENCE 调用 NEXTVAL 期间，应该以 SSX 模式获得 SQ 锁。许多会话同时为了获取 SQ 锁而发生争用过程中，若发生争用，则等待 enq: SQ - contention 事件。enq: SQ - contention 事件的 P2 值是 Sequence 的 OBJECT ID。因此，若利用 P2 值与 DBA_OBJECTS 的结合，就可以知道对哪个 SEQUENCE 发生了等待现象。

③ **SV 锁**:RAC 上节点之间顺序得到保障的情况下，调用 SEQUENCE.NEXTVAL 期间拥有。赋予 **CACHE + ORDER** 属性的 SEQUENCE 上发生，等待事件为 **DFS lock handle，解决办法为：尽量设置为 NOORDER 并增大其 CACHE 值。**

根据创建 Sequence 时赋予的属性，整理等待事件的结果如下：

- **NOCACHE: row cache lock**
- **CACHE + NOORDER: enq: SQ - contention**
- **CACHE + ORDER(RAC): DFS lock handle**

创建 SEQUENCE 赋予的 CACHE 值较小时，有 enq: SQ - contention 等待增加的趋势。CACHE 值较小时，内存上事先 CACHE 的值很快被耗尽，这时需要将数据字典信息物理修改后，再次执行 CACHE 的工作。在此期间，因

为一直拥有 SQ 锁，相应的 enq: SQ - contention 事件的等待时间也会延长。很不幸的是，在创建 SEQUENCE 时，将 CACHE 值的缺省值设定为较小的 20。因此创建使用量多的 SEQUENCE 时，CACHE 值应该取 1000 以上的较大值。

另外，偶尔一次性同时创建许多会话时，有时会发生 enq: SQ - contention 等待事件。其理由是 V$SESSION.AUDSID（auditing session id）列值是利用 Sequence 创建的。Oracle 在创建新的会话后，利用名为 SYS.AUDSES$的 Sequence 的 nextval，创建 AUDSID 值。SYS.AUDSES$ Sequence 的 CACHE 大小的缺省值设定为 20。许多会话同时连接时，可以将 SYS.AUDSES$ Sequence 的 CACHE 大小扩大至 1000，以此可以解决 enq: SQ - contention 等待问题。 10g 下默认 20，11g 下默认为 10000，通过如下的 SQL 可以查询：

```
SELECT * FROM dba_sequences d WHERE d.sequence_name ='AUDSES$';
```

	SEQUENCE_OWNER	SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	CYCLE_FLAG	ORDER_FLAG	CACHE_SIZE	LAST_NUMBER
1	SYS	AUDSES$	1	2000000000	1	Y	N	10000	892977

RAC 上创建 SEQUENCE 时，在赋予了 CACHE 属性的状态下，若没有赋予 ORDER 属性，则各节点将会把不同范围的 SEQUENCE 值 CACHE 到内存上。比如，拥有两个节点的 RAC 环境下，创建 CACHE 值为 100 的 SEQUENCE 时，1 号节点使用 1～100，2 号节点使用 101～200。若两个节点之间都通过递增方式使用 SEQUENCE，必须赋予如下 ORDER 属性。

```
SQL> CREATE SEQUENCE ORDERED_SEQUENCE CACHE 100 ORDER;
```

如果是已赋予了 CACHE+ORDER 属性的 SEQUENCE，Oracle 使用 SV 锁进行行同步。即，对赋予了 ORDER 属性的 Sequence 调用 nextval 时，应该以 SSX 模式拥有 SV 锁。在获取 SV 锁过程中，如果发生争用时，不是等待 row cache lock 事件或 enq: SQ - contention 事件，而是等待名为 DFS lock handle 事件。正因如此，V$EVENT_NAME 视图上不存在类似"enq:SV-contention"的事件。DFS lock handle 事件是在 OPS 或 RAC 环境下，除了高速缓冲区同步之外，还有行高速缓冲区或库高速缓冲区的为了同步获取锁的过程中等待的事件。若要保障多个节点之间 Sequence 顺序，应该在全局范围内获得锁，在此过程中会发生 DFS lock handle 等待。在获取 SV 锁的过程中发生的 DFS lock handle 等待事件的 P1 、P2 值与 enq: SQ - contention 等待事件相同（ P1=mode+namespace、P2=object# ）。因此从 P1 值能确认是否是 SV 锁，通过 P2 值可以确认对哪些 Sequence 发生过等待。SV 锁争用问题发生时的解决方法与 SQ 锁的情况相同，就是将 CACHE 值进行适当调整，这也是唯一的

方法。

在 RAC 等多节点环境下，Sequence 的 CACHE 值给性能带来的影响比单节点环境更严重。因此，尽量赋予 CACHE+NOORDER 属性，并要给予足够大的 CACHE 值。如果需要保障顺序，必须赋予 CACHE+ORDER 属性。但这时为了保障顺序，实例之间不断发生数据的交换。因此，与赋予了 NOORODER 属性的时候相比性能稍差。

有一点必须要注意 ,没有赋予 CACHE 属性时 ,不管 ORDER 属性使用与否或 RAC 环境与否 ,一直等待 row cache lock 事件。row cache lock 是可以在全局范围内使用的锁，单实例环境或多实例环境同样可以发生。

没有赋予 CACHE 属性时，不管 ORDER 属性是否或 RAC 环境是否，一直等待 ROW CACHE 事件，ROW CACHE LOCK 是否可以在全局范围内使用的锁，单实例环境或多实例环境同时可以发生。

Oracle Sequence 默认是 NOORDER，如果设置为 ORDER；在单实例环境没有影响，在 RAC 环境此时，多实例实际缓存相同的序列，此时在多个实例并发取该序列的时候，会有短暂的资源竞争来在多实例之间进行同步。因次性能相比 noorder 要差，所以 RAC 环境非必须的情况下不要使用 ORDER，尤其要避免 NOCACHE ORDER 组合。

但是如果使用了 Cache，如果此时 DB 崩溃了，那么 sequence 会从 cache 之后重新开始，在 cache 中没有使用的 sequence 会被跳过。即 sequence 不连续。所以只有在多节点高峰并发量很大的情况且对连续性要求不高的情况下，才使用：noorder + cache。

DFS lock handle

The session waits for the lock handle of a global lock request. The lock handle identifies a global lock. With this lock handle, other operations can be performed on this global lock (to identify the global lock in future operations such as conversions or release). The global lock is maintained by the DLM.

Wait Time: The session waits in a loop until it has obtained the lock handle from the DLM. Inside the loop there is a wait of 0.5 seconds.

Parameter	Description
name	See "name and type"
mode	See "mode"
id1	See "id1"
id2	See "id2"

The session needs to get the lock handle.

该等待事件的发生，若不是 SV 锁的话，多半为 bug 引起。

id1

The first identifier (id1) of the enqueue or global lock takes its value from P2 or P2RAW. The meaning of the identifier depends on the name (P1).

id2

The second identifier (id2) of the enqueue or global lock takes its value from P3 or P3RAW. The meaning of the identifier depends on the name (P1).

mode

The mode is usually stored in the low order bytes of P1 or P1RAW and indicates the mode of the enqueue or global lock request.This parameter has one of the following values:

Table C-1 Lock Mode Values

Mode Value	Description
1	Null mode
2	Sub-Share
3	Sub-Exclusive
4	Share
5	Share/Sub-Exclusive
6	Exclusive

Use the following SQL statement to retrieve the name of the lock and the mode of the lock request:

```
select chr(bitand(p1,-16777216)/16777215)||
chr(bitand(p1, 16711680)/65535) "Lock",
bitand(p1, 65535) "Mode"
from v$session_wait
where event = 'DFS enqueue lock acquisition';
```

name and type

The name or "type" of the enqueue or globallock can be determined by looking at the two high order bytes of P1 or P1RAW. The name is always two characters. Use the following SQL statement to retrieve the lock name.

```
select chr(bitand(p1,-16777216)/16777215)||
chr(bitand(p1,16711680)/65535) "Lock"
from v$session_wait
where event = 'DFS enqueue lock acquisition';
```

==========

有了以上的知识，我们知道，目前只需要找到产生等待的序列名称，然后设置其 CACHE 为比较大的一个值即可解决问题。

1.3.4 故障解决过程

1.3.4.1 enq: SQ - contention 等待事件

我们查询出现问题时间段的 ASH 视图 DBA_HIST_ACTIVE_SESS_HISTORY 来找到我们需要的序列名称。

可以有多种查询方法：

```
SELECT D.SQL_ID, COUNT(1)
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('20160823170000', 'YYYYMMDDHH24MISS') AND
       TO_DATE('20160823230000', 'YYYYMMDDHH24MISS')
   AND D.EVENT = 'enq: SQ - contention'
 GROUP BY D.SQL_ID;
```

	SQL_ID	COUNT(1)
3	d8b9vwzcws66b	222
6	b0wc3mqwqqd1d	222
5	66cawus5vz9qd	276
1	3yyf56pc9kzk3	786
4	dqwh46a2b7nr9	874
2	3jhvjgj7kbpmt	15426

可以看到 SQL_ID 为 3jhvjgj7kbpmt 的 SQL 最多，我们查看具体 SQL 内容：

```
SELECT * FROM V$SQL A WHERE A.SQL_ID IN ('3jhvjgj7kbpmt') ;
```

SQL_TEXT	SQL_FULLTEXT	SQL_ID
select ONLNID.nextval into :b1 from DUAL	<CLOB>	3jhvjgj7kbpmt

由此可以知道，产生等待的序列名称为 ONLNID,另外，我们也可以从 DBA_HIST_ACTIVE_SESS_HISTORY 视图的 P2 值获取到序列的名称，如下：

```
SELECT D.EVENT,
       D.P1TEXT,
       D.P1,
       D.P2TEXT,
       D.P2,
       CHR(BITAND(P1, -16777216) / 16777215) ||
       CHR(BITAND(P1, 16711680) / 65535) "Lock",
       BITAND(P1, 65535) "Mode",
       D.BLOCKING_SESSION,
       D.BLOCKING_SESSION_STATUS,
       D.BLOCKING_SESSION_SERIAL#,
       D.SQL_ID,
       TO_CHAR(D.SAMPLE_TIME, 'YYYYMMDDHH24MISS') SAMPLE_TIME,
       D.*
```

```
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('20160823170000', 'YYYYMMDDHH24MISS') AND
       TO_DATE('20160823230000', 'YYYYMMDDHH24MISS')
   AND D.EVENT = 'enq: SQ - contention';
```

EVENT	P1TEXT	P1	P2TEXT	P2	Lock	Mode	BLOCKING_SESSION
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	560
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	303
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	248
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	581
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	581
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	581
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	581
enq: SQ - contention	name|mode	1397817350	object #	47989	SQ	6	581

由以上的查询结果可知，序列的 object_id 为 47989，由此也可以知道序列名称如下，另外，lock 为 SQ 代表的是序列的 cache 锁（Sequence Cache），mode 为 6 代表 Exclusive 排他锁。

```
SELECT * FROM DBA_OBJECTS D WHERE D.object_id='47989';
```

OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE
ONLNID		47989		SEQUENCE

知道了序列名称后，我们就可以查询序列的属性了：

```
SELECT * FROM DBA_SEQUENCES D WHERE D.sequence_name='ONLNID' ;
```

Row 1	Fields	Comments
SEQUENCE_OWNER	NFXS	*Name of the owner of the sequence*
SEQUENCE_NAME	ONLNID	*SEQUENCE name*
MIN_VALUE	1	*Minimum value of the sequence*
MAX_VALUE	99999999998	*Maximum value of the sequence*
INCREMENT_BY	1	*Value by which sequence is incremented*
CYCLE_FLAG	Y	*Does sequence wrap around on reaching limit?*
ORDER_FLAG	N	*Are sequence numbers generated in order?*
CACHE_SIZE	20	*Number of sequence numbers to cache*
LAST_NUMBER	147439993	*Last sequence number written to disk*

可以看到，该序列是 NOORDER 属性，CACHE 值为默认的 20，对于并发值很高的系统而言，该默认值太低，所以需要调整到 1000，我们执行 SQL:ALTER SEQUENCE NFXS.ONLNID CACHE 1000; 调整其 cache 值即可解决该问题。

1.3.4.2 DFS lock handle 等待事件

我们查询出现问题时间段的 ASH 视图 DBA_HIST_ACTIVE_SESS_HISTORY 来找到我们需要的序列名称。

可以有多种查询方法：

```
SELECT D.SQL_ID, COUNT(1)
```

```
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('20160823170000', 'YYYYMMDDHH24MISS') AND
       TO_DATE('20160823230000', 'YYYYMMDDHH24MISS')
   AND D.EVENT = 'DFS lock handle'
 GROUP BY D.SQL_ID;
```

SQL_ID	COUNT(1)
67vjwqswg2zvy	12321
3c7n0kgnfsq2w	164
79dd2suvszkcz	68
fwumwrx6s058m	55
5cv358q9244cm	50
2dnnggav1p6su	50
ga09ss1bsh88w	46
8f4agkbax0tpu	27
091jpza3ybn3d	25
12fmbts8ws66n	22

可以看到 SQL_ID 为"67vjwqswg2zvy"的 SQL 最多，我们查看具体 SQL 内容：

```
SELECT * FROM V$SQL A WHERE A.SQL_ID IN ('67vjwqswg2zvy') ;
```

SQL_TEXT	SQL_FULLTEXT	SQL_ID	SHARABLE_MEM
SELECT formatid, globalid, branchid FROM SYS.D ···	<CLOB> ···	67vjwqswg2zvy	59400
SELECT formatid, globalid, branchid FROM SYS.DI ···	<CLOB> ···	67vjwqswg2zvy	51896
SELECT formatid, globalid, branchid FROM SYS.DI ···	<CLOB> ···	67vjwqswg2zvy	48680

SQL 的内容为：

```
SELECT formatid, globalid, branchid FROM                SYS.DBA_PENDING_TRANSACTIONS
ORDER BY formatid, globalid, branchid
```

很奇怪，这是个系统视图，为啥会有 DFS lock handle 的等待事件产生呢？

```
SELECT D.EVENT,
       D.P1TEXT,
       D.P1,
       D.P2TEXT,
       D.P2,
       CHR(BITAND(P1, -16777216) / 16777215) ||
       CHR(BITAND(P1, 16711680) / 65535) "Lock",
       BITAND(P1, 65535) "Mode",
       D.BLOCKING_SESSION,
       D.BLOCKING_SESSION_STATUS,
       D.BLOCKING_SESSION_SERIAL#,
       D.SQL_ID,
       TO_CHAR(D.SAMPLE_TIME, 'YYYYMMDDHH24MISS') SAMPLE_TIME,
       D.*
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('20160823170000', 'YYYYMMDDHH24MISS') AND
       TO_DATE('20160823230000', 'YYYYMMDDHH24MISS')
   AND D.EVENT = 'DFS lock handle';
```

EVENT	P1TEXT	P1	P2TEXT	P2	Lock	Mode	BLOCKING_SESSION	BLOCKING_SESSION_STATUS
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID
DFS lock handle …	type|mode …	1128857605	id1 …	10	CI	5	803	VALID

由以上的查询结果可知，lock 为 CI 代表的是交叉实例功能调用实例，而并不是我们期望的 SV 锁，mode 为 5 代表 Share/Sub-Exclusive。

```
SELECT * FROM V$LOCK_TYPE D WHERE D.TYPE ='CI';
```

TYPE	NAME	ID1_TAG	ID2_TAG	IS_USER	DESCRIPTION
CI …	Cross-Instance Call Invocation …	opcode …	type …	NO	Coordinates cross-instance function invocations …

查了 metalink 可知，该问题是由 bug 引起。该库的版本为 10.2.0.5 的基础版本，并没有打任何的 PSU。

1.3.5 健康检查

DB_healthcheck_by_lhr_22.188.188.72_ORANFXS_2_10.2.0.5.0_20160824173105.html

AWR 分析完成后，我又收集了一下该库的健康检查情况，看看是否有其它方面的问题。

数据库基本信息

巡检报告文件名称	DB_healthcheck_by_lhr_22.188.188.72_[redacted]_2_10.2.0.5.0_20160824173105.html
巡检时间	2016-08-24 (Wednesday) 17:29:55 PM timezone +08:00
当前巡检用户	SYS
当前巡检会话	INST_ID: 2, 【466,31571,3379374】
数据库服务器名称及IP地址	【[redacted]1: 22.188.188.71】,【[redacted]: 22.188.188.72】
数据库服务器配置情况	【Inst_id 1: CPUs:24 Cores:12 Sockets: Memory:24G】,【Inst_id 2: CPUs:24 Cores:12 Sockets: Memory:24G】
操作系统信息	AIX-Based Systems (64-bit) / 6
数据库名称	[redacted]
数据库全局名	[redacted]
当前实例名	[redacted]S2
所有实例名	[redacted]
数据库版本	10.2.0.5.0
数据库ID(DBID)	1654173250
是否RAC集群及其节点数	TRUE : 2
数据库创建时间	2016-07-11 15:04:34
实例启动时间	【INST_ID 2: 2016-08-19 08:49:28】,【INST_ID 1: 2016-08-19 08:49:21】
ORACLE_HOME	/oracle/app/oracle/product/10.2.0/db
ORACLE_SID	[redacted]
TNS_ADMIN	
数据库归档模式	ARCHIVELOG
数据库闪回状态	NO
数据库字符集	ZHS16CGB231280
数据库块大小	8192
强制日志	NO
数据库角色	PRIMARY
是否有DG	NO
是否有OGG	NO
db time zone	4
回收站情况	状态: on, 占用空间: M, 共0个对象
特殊表空间情况(G)	SYSAUX:1/4,SYSTEM:0/4,TEMP:0/15,UNDOTBS1:20/20,UNDOTBS2:20/20
数据库大小	All TS Info:【ts_size: 563.97G , Used_Size: 150.14G , Used_per: 26.62% , MAX_Size: 564G】

• 数据库系统PSU信息

可以看出这个库并没有任何的 PSU 的信息，然后我们直接查看检查的结果：

健康检查结果			
健康检查结果	健康检查结果	健康检查过程中脚本产生的错误	

健康检查报告结果

ID	WARING_LEVEL	CHECK_TYPE	CHECK_MESSAGE	CHECK_MESSAGE_DETAIL_LINK
1	2	巡检服务明细.RMAN信息	数据库无RMAN备份信息，强烈建议对数据库进行备份	[参考：RMAN信息]
2	4	数据库对象.无效对象	数据库里有无效的对象，建议重新编译	[参考：无效对象]
3	2	数据库对象.其他对象.告警日志	数据库告警日志有ora错误，请详细检查告警日志内容	[参考：告警日志]
4	2	数据库对象.其他对象.序列cache小于20	数据库序列的cache值小于20，可能伴随有enq: SQ - contention等待事件	[序列cache小于20]
5	2	数据库性能分析.会话.超过10小时无响应的会话	超过10小时无响应的会话可以考虑将其kill掉来释放资源	[参考：超过10小时无响应的会话]

[回到目录][健康检查过程中脚本产生的错误]

可以看到数据库有上边的几个问题，其中就有 cache 小于 20 的问题，我们点击连接可以看到：

• 历史等待事件中是否有序列等待，该部分的序列强烈建议修改其cache值

• 所有序列等待总况

EVENT	P2	USERNAME	SEQUENCE_NAME	SEQUENCE_CACHE	'ALTERSEQUENCE'||WB.USERN
enq: SQ - contention	47994	NFXS	TRANID	1000	ALTER SEQUENCE NFXS.TRANID CACHE 1000;
enq: SQ - contention	47991	NFXS	RALGID	1000	ALTER SEQUENCE NFXS.RALGID CACHE 1000;
enq: SQ - contention	47986	NFXS	MDELID	1000	ALTER SEQUENCE NFXS.MDELID CACHE 1000;
enq: SQ - contention	47989	NFXS	ONLNID	1000	ALTER SEQUENCE NFXS.ONLNID CACHE 1000;
enq: SQ - contention	47993	NFXS	SPOLID	1000	ALTER SEQUENCE NFXS.SPOLID CACHE 1000;
enq: SQ - contention	47992	NFXS	RDELID	1000	ALTER SEQUENCE NFXS.RDELID CACHE 1000;

另外，在告警日志中，我们也可以看到，如下的信息，说明 prcesses 参数设置过小。

- 查看近一月内最新的10条ora告警日志记录，按照时间倒序排列

ID	INST_ID	alert_date	message_text
10266	1		ORA-00018: maximum number of sessions exceeded
10259	1		ORA-00018: maximum number of sessions exceeded
10258	1		ORA-00604: error occurred at recursive SQL level 1
10111	1		ORA-00018: maximum number of sessions exceeded
10029	1		ORA-00018: maximum number of sessions exceeded
9971	1		ORA-00018: maximum number of sessions exceeded
9817	1		ORA-00018: maximum number of sessions exceeded
9533	1		ORA-00018: maximum number of sessions exceeded
9532	1		ORA-00604: error occurred at recursive SQL level 1
9531	1		ORA-01001: invalid cursor
63090	2		ORA-00018: maximum number of sessions exceeded
63089	2		ORA-00604: error occurred at recursive SQL level 1
63086	2		ORA-00018: maximum number of sessions exceeded
63079	2		ORA-00018: maximum number of sessions exceeded
63078	2		ORA-00604: error occurred at recursive SQL level 1
63061	2		ORA-00018: maximum number of sessions exceeded
63060	2		ORA-00604: error occurred at recursive SQL level 1
63049	2		ORA-00018: maximum number of sessions exceeded
63048	2		ORA-00604: error occurred at recursive SQL level 1
63036	2		ORA-00018: maximum number of sessions exceeded

About Me

- 本文作者：小麦苗，只专注于数据库的技术，更注重技术的运用
- 本文在 itpub（http://blog.itpub.net/26736162）、博客园(http://www.cnblogs.com/lhrbest)和个人微信公众号（xiaomaimiaolhr）上有同步更新，推荐 pdf 文件阅读
- QQ 群：230161599 微信群：私聊
- 本文 itpub 地址：http://blog.itpub.net/26736162/viewspace-2123996/ 博客园地址：http://www.cnblogs.com/lhrbest/articles/5804363.html
- 本文 pdf 版：http://yunpan.cn/cdEQedhCs2kFz （提取码：ed9b）
- 小麦苗分享的其它资料：http://blog.itpub.net/26736162/viewspace-1624453/
- 联系我请加 QQ 好友(642808185)，注明添加缘由
- 于 2016-08-24 09:00~2016-08-24 19:00 在中行完成

长按识别二维码或微信客户端扫描下边的二维码来关注小麦苗的微信公众号：xiaomaimiaolhr,学习最实用的数据库技术。