【故障处理】队列等待之 enq: US - contention 案例

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: US - contention 等待事件的解决

② 一般等待事件的解决办法

③ 队列等待的基本知识

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	11.2.0.4.0
db 存储	ASM

OS 版本及 kernel 版本	AIX 64 位 7.1.0.0

1.3.2 故障发生现象及报错信息

最近系统做压测，碰到的问题比较多，今天同事发了个 AWR 报告，说是系统响应很慢，我简单看了下，简单分析下吧：

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
[redacted]	3860591551	ora[redacted]	2	29-Jul-16 15:07	11.2.0.4.0	YES

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
[redacted]	AIX-Based Systems (64-bit)	32	8		48.00

	Snap Id	Snap Time	Sessions	Cursors/Session	Instances
Begin Snap:	1445	07-Sep-16 17:39:44	246	4.4	2
End Snap:	1451	07-Sep-16 22:13:41	1473	7.7	2
Elapsed:		273.95 (mins)			
DB Time:		2,009.96 (mins)			

270 分钟时间而 DB Time 为 2000 多分钟，DB Time 太高了，负载很大，很可能有异常的等待事件，系统配置还是比较牛逼的。

Report Summary

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	7.3	0.0	0.00	0.00
DB CPU(s):	0.5	0.0	0.00	0.00
Redo size (bytes):	893,305.1	1,277.1		
Logical read (blocks):	13,262.5	19.0		
Block changes:	4,586.6	6.6		
Physical read (blocks):	413.8	0.6		
Physical write (blocks):	158.6	0.2		
Read IO requests:	147.3	0.2		
Write IO requests:	84.0	0.1		
Read IO (MB):	3.2	0.0		
Write IO (MB):	1.2	0.0		
Global Cache blocks received:	961.2	1.4		
Global Cache blocks served:	940.8	1.3		
User calls:	5,505.2	7.9		
Parses (SQL):	781.5	1.1		
Hard parses (SQL):	0.2	0.0		
SQL Work Area (MB):	0.6	0.0		
Logons:	0.1	0.0		
Executes (SQL):	4,196.1	6.0		
Rollbacks:	0.0	0.0		
Transactions:	699.5			

事务量很大，其它个别参数有点问题，不一一解说了。等待事件很明显了：

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
enq: US - contention	1,190,235	52.7K	44	43.7	Other
row cache lock	1,446,985	18.1K	12	15.0	Concurrency
log file sync	11,498,850	14.1K	1	11.7	Commit
DB CPU		8860		7.3	
gc cr block 2-way	9,444,401	5958.3	1	4.9	Cluster
gc current block 2-way	5,591,598	3462.8	1	2.9	Cluster
gc buffer busy acquire	378,851	3397.7	9	2.8	Cluster
db file sequential read	1,508,135	2978.4	2	2.5	User I/O
enq: TA - contention	1,390	2882.3	2074	2.4	Other
gc buffer busy release	44,775	1124.2	25	.9	Cluster

AWR 的其它部分就不分析了，首先这个等待事件：enq: US - contention 比较少见，查了一下资料，有点收获：

```
SELECT * FROM V$EVENT_NAME WHERE NAME = 'enq: US - contention';
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
779	2458904239	enq: US - contention	name|mode	undo segment #	0	1893977003	0	Other

```
SELECT * FROM v$lock_type d WHERE d.TYPE='US';
```

TYPE	NAME	ID1_TAG	ID2_TAG	IS_USER	DESCRIPTION
US	Undo Segment	undo segment #	0	NO	Lock held to perform DDL on the undo segment

"enq: US - contention"，这个 event 说明事务在队列中等待 UNDO Segment，通常是由于 UNDO 空间不足导致的。

在对此事件说明之前，需要理解在使用 AUM（atuomatic undo management）时，回滚段在何时联机或脱机。AUM 与 RBU（rollback segment management）不同，回滚段的管理是 Oracle 自动完成的。使用 AUM 时，回滚段的联机或脱机的时刻如下：

1）在执行 alter database open 的时候将回滚段联机

2）通过 alter system set undo_tablespace=xxx 修改撤销表空间时，将原来的回滚段脱机后，再将新的回滚段联机。

3）通过 SMON，自动脱机或者联机回滚段，如果一段时间内，事务量增加，联机状态的回滚段也会增加，一段时间内若是没有实物或事务减少，回滚段就会被 smon 进程脱机。

为了同步将回滚段联机或脱机的过程，执行该工作的服务器进程或后台进程应获得 US 锁，每个回滚段非配一个 US 锁，ID1=Undo segment#。若在获得 US 锁的过程中发生争用，则等待 enq：US-contention 事件。服务器进程应该在开始事务时分配到回滚段，但如果不存在可用的回滚段时，应该创建新的回滚段或将脱机状态的回滚段联机。在实现此项工作期间，服务器进程为了获得 US 锁而等待，等待占有可用回滚段。

这是 oracle10g 中开始出现的 bug(在 11.1.0.7 中仍有这个 BUG)，当因为系统 activity 增加或者降低的时候，oracle SMON 进程会自动 ONLINE 或者 OFFLINE rollback segments。这样导致某些与 undo segments 相关的 latch 或者 enqueue 被 hold 住太长时间 ,导致系统很多活跃 session 都开始等待 enq: US - contention。可以同时使用以下解决方法：

1．设置 event 让 SMON 不自动 OFFLINE 回滚段

```
alter system set events '10511 trace name context forever, level 1';
```

2．设置参数_rollback_segment_count ：表示有多少 rollback segment 要处于 online 的状态；可以将该数值设置为数据库最繁忙的时候的回滚段数目。

```
alter system set "_rollback_segment_count"=1000 SID='*';
```

这里以"_"开头的为隐藏参数，通过 show parameter 是看不到的，可以通过以下语句：

```
select a.ksppinm name, b.ksppstvl value, a.ksppdesc description
from x$ksppi a, x$ksppcv b
where a.indx = b.indx
and a.ksppinm like '%_rollback_segment_count%';
```

3. undo autotune bug 多多。最好 disable。

```
alter system set "_undo_autotune"= false;
```

这种方法就是关闭了 UNDO 的自动调整功能，同时也能解决掉 UNDO 表空间会在很长时间都一直保持着使用率是接近 100%的问题。

4. 有一个 patch: A fix to bug 7291739 is to set a new hidden parameter, _highthreshold_undoretention to set a high threshold for undo retention completely distinct from maxquerylen.

```
alter system set "_highthreshold_undoretention"=;
```

5. 增加 undo 表空间

```
alter tablespace UNDOTBS1 add datafile '+DATA1' size 30G;
```

6. 设置 undo 表空间的 NOGUARANTEE

```
select tablespace_name, retention from dba_tablespaces where tablespace_name like 'UNDO%';
ALTER TABLESPACE UNDOTBS RETENTION NOGUARANTEE;
```

7. 减少 UNDO_RETENTION 的时间

```
SQL> show parameter undo
NAME                                 TYPE        VALUE
------------------ ---------------------- --------
undo_management                      string       AUTO
undo_retention                       integer      10800
```

8. 重启数据库节点

1.3.3 故障分析及解决

我们查询 ASH 视图看看当时的情况：

```
SELECT D.SQL_ID,CHR(BITAND(P1, -16777216) / 16777215) ||
```

```
       CHR(BITAND(P1, 16711680) / 65535) "Lock",
       BITAND(P1, 65535) "Mode", COUNT(1),COUNT(DISTINCT d.session_id )
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('2016-09-07 17:39:44', 'YYYY-MM-DD
HH24:MI:SS') AND
       TO_DATE('2016-09-07 22:13:41', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: US - contention'
 GROUP BY D.SQL_ID,(CHR(BITAND(P1, -16777216) / 16777215) ||
       CHR(BITAND(P1, 16711680) / 65535)),(BITAND(P1, 65535));
```

	SQL_ID	Lock	Mode	COUNT(1)	COUNT(DISTINCTD.SESSION_ID)
1		US	6	9	1
2	26ad9zvt5xgb3	US	6	5066	935
3	1cmnjddakrqbv	US	6	1184	317
4	5ww8x9u15a90y	US	6	41	16
5	ayngk81z8fh0m	US	6	564	133

LOCK 为 US，MODE 为 6，看看具体的 SQL 内容：

```
 SELECT A.SQL_TEXT, A.EXECUTIONS, A.MODULE, A.SQL_ID
   FROM V$SQL A
  WHERE A.SQL_ID IN ('5ww8x9u15a90y',
                     'ayngk81z8fh0m',
                     '1cmnjddakrqbv',
                     '26ad9zvt5xgb3');
```

	SQL_TEXT	EXECUTIONS	MODULE	SQL_ID
9	insert into trans_book_success (TRANS_ID, PRO_ID, PF	1076592	JDBC Thin Client	5ww8x9u15a90y
10	insert into trans_book_success (TRANS_ID, PRO_ID, PF	1641222	JDBC Thin Client	5ww8x9u15a90y
6	insert into trans_book_success(trans_id,id_type,id_num,nan	4845078	JDBC Thin Client	26ad9zvt5xgb3
7	insert into trans_book_success(trans_id,id_type,id_num,nan	69	JDBC Thin Client	26ad9zvt5xgb3
8	insert into trans_book_success(trans_id,id_type,id_num,nan	5611552	JDBC Thin Client	26ad9zvt5xgb3
2	insert into trans_book_success(trans_id,id_type,id_num,nar	4575017	JDBC Thin Client	26ad9zvt5xgb3
1	insert into trans_book_success(trans_id,id_type,id_num,nan	431284	JDBC Thin Client	26ad9zvt5xgb3
3	insert into trans_book_success(trans_id,id_type,id_num,nan	317	JDBC Thin Client	26ad9zvt5xgb3
5	insert into trans_book_success(trans_id,id_type,id_num,nan	785	JDBC Thin Client	26ad9zvt5xgb3
4	insert into trans_book_success(trans_id,id_type,id_num,nan	1364139	JDBC Thin Client	26ad9zvt5xgb3
13	update organization o set o.quota_unused = o.quota_unuse	6024996	JDBC Thin Client	1cmnjddakrqbv
14	update organization o set o.quota_unused = o.quota_unuse	6729732	JDBC Thin Client	1cmnjddakrqbv
11	update trans_book_success b set b.book_count = :1,b.moc	450242	JDBC Thin Client	ayngk81z8fh0m
12	update trans_book_success b set b.book_count = :1,b.moc	768413	JDBC Thin Client	ayngk81z8fh0m

看看时间段是哪个区间：

```
 SELECT D.SQL_ID, TO_CHAR(D.SAMPLE_TIME, 'YYYY-MM-DD HH24:MI'), COUNT(1)
   FROM DBA_HIST_ACTIVE_SESS_HISTORY D
  WHERE D.EVENT = 'enq: US - contention'
    AND D.SQL_ID IN ('5ww8x9u15a90y', '26ad9zvt5xgb3')
  GROUP BY D.SQL_ID, TO_CHAR(D.SAMPLE_TIME, 'YYYY-MM-DD HH24:MI');
```

	SQL_ID	TO_CHAR(D.SAMPLE_TIME,'YYYY-MM	COUNT(1)
1	26ad9zvt5xgb3	2016-09-07 18:37	2896
2	5ww8x9u15a90y	2016-09-07 18:37	16
3	26ad9zvt5xgb3	2016-09-07 18:36	2170
4	5ww8x9u15a90y	2016-09-07 18:36	25

看来问题几种在这 2 分钟之内。基本都是对同一个表做插入或者更新操作：

```
SELECT DISTINCT D.CURRENT_OBJ#
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN
       TO_DATE('2016-09-07 17:39:44', 'YYYY-MM-DD HH24:MI:SS') AND
       TO_DATE('2016-09-07 22:13:41', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: US - contention'
 GROUP BY D.CURRENT_OBJ#;
SELECT * FROM DBA_OBJECTS a WHERE a.object_id IN
('87620','87632','87663','87667','87686','87684','87688','87626','87646','87642','87639','87661','87628','87675','87643','87677','87660','87631','87629','87668','87682','87685','87654','87640','87627','87636','87664','87655','87645','87637','87669','87673','87666','87634','87644','87672','87648','87649','87662','87651','87641','87653','87659','87680','87681','0','87625','87670','87658','87674','87671','87633','87679','87647');
```

	OWNER	OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE
4	[redacted]	TRANS_BOOK_SUCCESS	SYS_P43	87627	90180	TABLE PARTITION
5	[redacted]	TRANS_BOOK_SUCCESS	SYS_P44	87628	90181	TABLE PARTITION
6	[redacted]	TRANS_BOOK_SUCCESS	SYS_P45	87629	90182	TABLE PARTITION
7	[redacted]	TRANS_BOOK_SUCCESS	SYS_P47	87631	90184	TABLE PARTITION
8	[redacted]	TRANS_BOOK_SUCCESS	SYS_P48	87632	90185	TABLE PARTITION
9	[redacted]	TRANS_BOOK_SUCCESS	SYS_P49	87633	90186	TABLE PARTITION
10	[redacted]	TRANS_BOOK_SUCCESS	SYS_P50	87634	90187	TABLE PARTITION
11	[redacted]	TRANS_BOOK_SUCCESS	SYS_P52	87636	90189	TABLE PARTITION
12	[redacted]	TRANS_BOOK_SUCCESS	SYS_P53	87637	90190	TABLE PARTITION
13	[redacted]	TRANS_BOOK_SUCCESS	SYS_P55	87639	90192	TABLE PARTITION
14	[redacted]	TRANS_BOOK_SUCCESS	SYS_P56	87640	90193	TABLE PARTITION
15	[redacted]	TRANS_BOOK_SUCCESS	SYS_P57	87641	90194	TABLE PARTITION
16	[redacted]	TRANS_BOOK_SUCCESS	SYS_P58	87642	90195	TABLE PARTITION
17	[redacted]	TRANS_BOOK_SUCCESS	SYS_P59	87643	90196	TABLE PARTITION
18	[redacted]	TRANS_BOOK_SUCCESS	SYS_P60	87644	90197	TABLE PARTITION
19	[redacted]	TRANS_BOOK_SUCCESS	SYS_P61	87645	90198	TABLE PARTITION

可以看到操作的表是一个分区表。

解决方案：

```
alter tablespace UNDOTBS1 add datafile '+DATA1' size 30G;
alter system set events '10511 trace name context forever,level 1';
ALTER SYSTEM SET "_rollback_segment_count"=1000 SID='*';
```

执行之后经过开发进行压测，已经没有该等待事件的产生了：

```
SELECT D.SQL_ID, TO_CHAR(D.SAMPLE_TIME, 'YYYY-MM-DD HH24:MI'), COUNT(1)
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.EVENT = 'enq: US - contention'
 GROUP BY D.SQL_ID, TO_CHAR(D.SAMPLE_TIME, 'YYYY-MM-DD HH24:MI');
```

查询无数据。

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-2124767/ 博客园地址：http://www.cnblogs.com/lhrbest/articles/5858001.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 2016-09-08 09:00~2016-09-08 19:00 在中行完成
- 【版权所有，文章允许转载，但须以链接方式注明源地址，否则追究法律责任】

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。