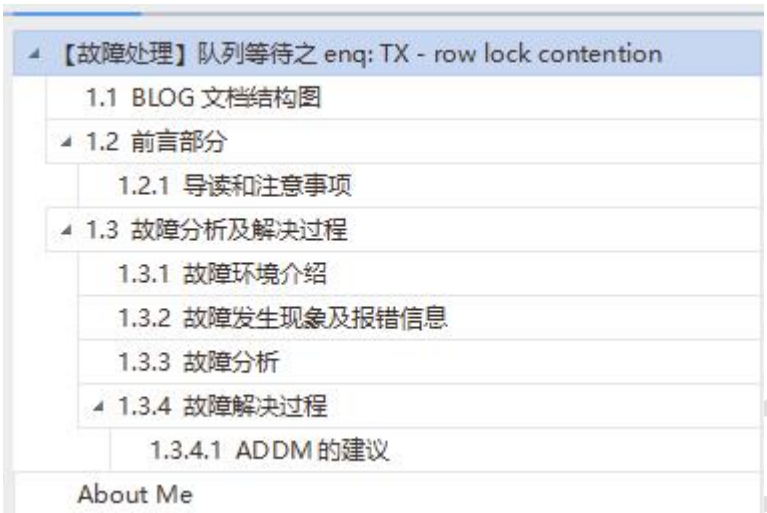
【故障处理】队列等待之 enq: TX - row lock contention

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: TX - row lock 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

④ ADDM 的使用

⑤ 如何获取历史执行计划

⑥ 查询绑定变量的具体值

⑦ 很多有用的查询性能的 SQL 语句

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

早上同事过来跟我说昨天有一套数据库做压力测试的时候，CPU 利用率很高，他已经抓取当时的 AWR，让我帮忙分析分析，下边我们来看看 AWR 中的数据：

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
[redacted]	3860591551	[redacted]	1	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:	1145	31-Aug-16 17:33:52	99	2.0	2
End Snap:	1148	31-Aug-16 18:14:36	588	6.5	2
Elapsed:		40.74 (mins)			
DB Time:		15,180.51 (mins)			

从 AWR 报告的头部可以分析得到，数据库为 RAC 库，11.2.0.4 版本，AIX64 位系统，有 32 颗 CPU，共 48G 内存，收集了 40 分钟内的 AWR 报告，但是 DB Time 有 15180 分钟，约为 15180/40=379 倍，说明这段时间内系统的负载异常的大。

如果关注数据库的性能，那么当拿到一份 AWR 报告的时候，最想知道的第一件事情可能就是系统资源的利用情况了，而首当其冲的，就是 CPU。而细分起来，CPU 可能指的是

- OS 级的 User%, Sys%, Idle%
- DB 所占 OS CPU 资源的 Busy%
- DB CPU 又可以分为前台所消耗的 CPU 和后台所消耗的 CPU

我们分析以下主机 CPU 的情况：

Host CPU

CPUs	Cores	Sockets	Load Average Begin	Load Average End	%User	%System	%WIO	%Idle
32	8		0.19	1.14	2.9	2.3	0.0	94.8

Instance CPU

%Total CPU	%Busy CPU	%DB time waiting for CPU (Resource Manager)
2.2	42.2	0.0

分析上面的主机图片，我们可以得出下面的结论：

- **OS 级的 User%，Sys%，Idle%：**

OS 级的%User 为 2.9，%Sys 为 2.3，%Idle 为 94.8，所以%Busy 应该是 100-94.8=5.2。

- **DB 所占 OS CPU 资源的 Busy%**

DB 占了 OS CPU 资源的 2.2，%Busy CPU 则可以通过上面的数据得到：

%Busy CPU = %Total CPU/(%Busy) * 100 = 2.2/5.2* 100 = 42.3，和报告的 42.2 相吻合。

接下来我们看看 Load Profile 部分：

Report Summary

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	372.7	1.0	0.17	0.13
DB CPU(s):	0.5	0.0	0.00	0.00
Redo size (bytes):	505,246.2	1,410.6		
Logical read (blocks):	12,149.6	33.9		
Block changes:	2,640.5	7.4		
Physical read (blocks):	2.3	0.0		
Physical write (blocks):	57.6	0.2		
Read IO requests:	2.3	0.0		
Write IO requests:	26.6	0.1		
Read IO (MB):	0.0	0.0		
Write IO (MB):	0.5	0.0		
Global Cache blocks received:	353.9	1.0		
Global Cache blocks served:	384.9	1.1		
User calls:	2,900.8	8.1		
Parses (SQL):	403.6	1.1		
Hard parses (SQL):	0.1	0.0		
SQL Work Area (MB):	0.7	0.0		
Logons:	0.5	0.0		
Executes (SQL):	2,145.0	6.0		
Rollbacks:	0.0	0.0		
Transactions:	358.2			

可以看到，每秒的事务数大约为 358，非常大，接下来看看等待事件：

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
enq: TX - row lock contention	3,813,533	855.8K	224	94.0	Application
gc buffer busy release	1,289,223	28.9K	22	3.2	Cluster
gc buffer busy acquire	351,771	13K	37	1.4	Cluster
buffer busy waits	2,460,151	7584	3	.8	Concurrency
latch: cache buffers chains	4,479,883	1565.2	0	.2	Concurrency
DB CPU		1293.2		.1	
log file sync	871,708	946.3	1	.1	Commit
gc current block busy	121,071	837.3	7	.1	Cluster
gc cr block 2-way	508,490	303.1	1	.0	Cluster
gc current block 2-way	226,716	131.7	1	.0	Cluster

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: TX - row lock contention 这个等待事件异常。Top 10 Foreground Events by Total Wait Time 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 10 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: TX - row lock contention **等待了 3,813,533 次，等待时间为 855,777 秒，平均等待时间为 855777/3813533=224 毫秒，占 DB Time 的 94%，等待类别为 Application 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: TX - row lock contention 这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个等待事件的知识。

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

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
241	310662678	enq: TX - row lock contention	name|mode	usn<<16 | slot	sequence	4217450380	1	Application

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

TYPE	NAME	ID1_TAG	ID2_TAG	IS_USER	DESCRIPTION
TX	Transaction	usn<<16 | slot	sequence	YES	Lock held by a transaction to allow other transactions to wait for it

```
SELECT D.EQ_NAME, D.EQ_TYPE, D.REQ_REASON, D.REQ_DESCRIPTION
      FROM V$ENQUEUE_STATISTICS D
     WHERE D.EQ_TYPE IN ('TX')
 AND D.REQ_REASON='row lock contention';
```

EQ_NAME	EQ_TYPE	REQ_REASON	REQ_DESCRIPTION
Transaction	TX	row lock contention	Lock held on a particular row by a transaction to prevent other transactions from modifying it

等待事件 enq: TX - row lock contention 中的 enq 是 enquence 的简写。enquence 是协调访问数据

库资源的内部锁。

所有以“enq:”打头的等待事件都表示这个会话正在等待另一个会话持有的内部锁释放，它的名称格式是enq:enqueue_type - related_details。这里的enqueue_type是TX，related_details是row lock contention。数据库动态性能视图v$event_name提供所有以“enq:”开头的等待事件的列表。

Oracle的enqueue包含以下模式：

模式代码	解释
1	Null mode
2	Sub-Share
3	Sub-Exclusive
4	Share
5	Share/Sub-Exclusive
6	Exclusive

enq: TX - row lock contention通常是Application级别的问题。通常情况下，Oracle数据库的等待事件enq: TX - row lock contention会在下列三种情况下会出现。

（一）第一种情况，是**真正的业务逻辑上的行锁冲突**，如一条记录被多个人同时修改。这种锁对应的请求模式是6（Waits for TX in mode 6 ：A会话持有row level lock，B会话等待这个lock释放。）。不同的session更新或删除同一个记录。（This occurs when one application is updating or deleting a row that another session is also trying to update or delete. ）

解决办法：持有锁的会话commit或者rollback。

（二）第二种情况，是**唯一键冲突（In mode 4,唯一索引）**，如主键字段相同的多条记录同时插入。这种锁对应的请求模式是4。这也是应用逻辑问题。表上存在唯一索引，A会话插入一个值（未提交），B会话随后也插入同样的值;A会话提交后，enq: TX - row lock contention消失。

解决办法：持有锁的会话commit或者rollback。

（三）第三种情况，是**bitmap索引的更新冲突（in mode 4 :bitmap）**，就是多个会话同时更新bitmap索引的同一个数据块。源于bitmap的特性：位图索引的一个键值，会指向多行记录，所以更新一行就会把该键值指向的所有行锁定。此时会话请求锁的对应的请求模式是4。bitmap索引的物理结构和普通索引一样，也是B-tree结

构,但它存储的数据记录的逻辑结构为"key_value,start_rowid,end_rowid,bitmap"。

其内容类似这样："‘8088’,00000000000,10000034441,10010001000011111000"

Bitmap 是一个二进制，表示 START_ROWID 到 END_ROWID 的记录，1 表示等于 key_value 即‘8088’的 ROWID 记录， 0 则表示不是这个记录。

解决办法：持有锁的会话 commit 或者 rollback。

在了解 bitmap 索引的结构之后，我们就能理解同时插入多条记录到拥有 bitmap 索引的表时，就会同时更新 bitmap 索引中一个块中的记录，等于某一个记录被同时更新，自然就会出现行锁等待。插入并发量越大，等待越严重。

（四）其他原因

It could be a primary key problem; a trigger firing attempting to insert, delete, or update a row; a problem with initrans; waiting for an index split to complete; problems with bitmap indexes;updating a row already updated by another session; or something else.

（https://forums.oracle.com/forums/thread.jspa?threadID=860488）

如果数据库一出现 enq: TX - row lock contention 等待，可以去看 v$session 和 v$session_wait 等视图。在 v$session 和 v$session_wait 中，如果看到的 event 列是 enq: TX - row lock contention 的,就表示这个会话正处于行锁等待。该等待事件的请求模式可以从 v$session 和 v$session_wait 的 p1 列中得到。

```
select sid,
       chr(bitand(p1, -16777216) / 16777215) ||
       chr(bitand(p1, 16711680) / 65535) "Name",
       (bitand(p1, 65535)) "Mode"
  from v$session_wait
 where event like 'enq%';
```

通过这个 SQL 可以将 p1 转换为易阅读的文字。

==========

有了以上的知识，我们知道，目前首先需要找到产生等待事件的类型进而来分析解决问题。

1.3.4 故障解决过程

根据 AWR 报告可以得到故障的时间是'2016-08-31 17:30:00'到'2016-08-31 18:15:00'之间。

我们查询出现问题时间段的 ASH 视图 DBA_HIST_ACTIVE_SESS_HISTORY 来找到我们需要的锁类型及 SQL 语句。

```
SELECT D.SQL_ID, COUNT(1)
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('2016-08-31 17:30:00', 'YYYY-MM-DD
HH24:MI:SS') AND
       TO_DATE('2016-08-31 18:15:00', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: TX - row lock contention'
 GROUP BY D.SQL_ID;
```

SQL_ID	COUNT(1)
1cmnjddakrqbv	176197

只有一条 SQL 语句，看来就是它了，我们来看看锁的类型：

```
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('2016-08-31 17:30:00', 'YYYY-MM-DD
HH24:MI:SS') AND
       TO_DATE('2016-08-31 18:15:00', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: TX - row lock 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)
1cmnjddakrqbv	TX	6	176197	556

看来约有 556 个会话等待该锁，锁为 TX 锁，模式为 6，刚好是我们之前的分析的原因中的第一种（第一种情况，是**真正的业务逻辑上的行锁冲突**，如一条记录被多个人同时修改。这种锁对应的请求模式是 6）。我们可以分析具体的对象是哪个：

```
SELECT  D.current_obj#,
        D.current_file#,
        D.current_block#,
        D.current_row#,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
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN TO_DATE('2016-08-31 17:30:00', 'YYYY-MM-DD
HH24:MI:SS') AND
       TO_DATE('2016-08-31 18:15:00', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: TX - row lock contention';
```

CURRENT_OBJ#	CURRENT_FILE#	CURRENT_BLOCK#	CURRENT_ROW#	EVENT	P1TEXT	P1	P2TEXT	P2
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	8
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	1
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	8
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	7
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	7
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	8
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	7
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	7
87620	14	171	23	enq: TX - row lock contention	name|mode	1415053318	usn<<16 | slot	7

```
SELECT * FROM dba_objects D WHERE D.object_id=87620;
```

	OWNER	OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE	CREATED
1	CNSL	ORGANIZATION		87620	87620	TABLE	2016/8/4 11:34:45

可以看到等待的是一张表。

可以看到 SQL_ID 为 1cmnjddakrqbv 的 SQL 最多，我们查看具体 SQL 内容：

```
SELECT A.* FROM V$SQL A WHERE A.SQL_ID IN ('1cmnjddakrqbv') ;
```

SQL_TEXT	SQL_FULLTEXT	SQL_ID
update organization o set o.quota_unused = o.quota_unus	<CLOB>	1cmnjddakrqbv

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

SQL_TEXT	EXECUTIONS	MODULE
update organization o set o.quota_unused = o.quota_unus	1443418	JDBC Thin Client

可以看到，该 SQL 是 JDBC 的，也就是 JAVA 前台的语句，我们将 SQL 语句拷贝出来：

```
update organization o set o.quota_unused = o.quota_unused-:1,o.mod_date =
systimestamp where o.quota_unused >= :2 and o.bank_id=:3 and o.pro_id=:4
```

SQL 语句是一个 UPDATE 语句，我们查询其执行计划是否正确，是否有索引：

```
SELECT *
  FROM TABLE(DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_HTML(L_DBID     => 3860591551,
                                                          L_INST_NUM => 1,
                                                          L_BID      => 1145,
                                                          L_EID      => 1148,
                                                          L_SQLID    => '1cmnjddakrqbv'));
```

	OUTPUT
1	<html lang="en"><head><title>AWR SQL Report for DB: ORACNSL, Inst: oraCNSL…
2	<style type="text/css"> …
3	body.awr {font:bold 10pt Arial,Helvetica,Geneva,sans-serif;color:black; backgroun…
4	pre.awr {font:8pt Courier;color:black; background:White;} …
5	h1.awr {font:bold 20pt Arial,Helvetica,Geneva,sans-serif;color:#336699;backgrou…
6	h2.awr {font:bold 18pt Arial,Helvetica,Geneva,sans-serif;color:#336699;backgrou…
7	h3.awr {font:bold 16pt Arial,Helvetica,Geneva,sans-serif;color:#336699;backgrour…
8	li.awr {font: 8pt Arial,Helvetica,Geneva,sans-serif; color:black; background:White;}…
9	th.awrnobg {font:bold 8pt Arial,Helvetica,Geneva,sans-serif; color:black; backgrou…
10	th.awrbg {font:bold 8pt Arial,Helvetica,Geneva,sans-serif; color:White; backgroun…

拷贝到 txt 中另存为 html 即可以看到报告：

Plan Statistics

- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Stat Name	Statement Total	Per Execution	% Snap Total
Elapsed Time (ms)	907,663,405	2,404.87	99.65
CPU Time (ms)	675,633	1.79	52.24
Executions	377,427		
Buffer Gets	17,095,307	45.29	57.57
Disk Reads	0	0.00	0.00
Parse Calls	377,248	1.00	38.25
Rows	377,412	1.00	
User I/O Wait Time (ms)	603		
Cluster Wait Time (ms)	42,838,883		
Application Wait Time (ms)	855,395,359		
Concurrency Wait Time (ms)	9,080,681		
Invalidations	0		
Version Count	26		
Sharable Mem(KB)	89		

Back to Plan 1(PHV: 374610298)
Back to Top

Execution Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	UPDATE STATEMENT				2 (100)	
1	UPDATE	ORGANIZATION				
2	TABLE ACCESS BY INDEX ROWID	ORGANIZATION	1	30	2 (0)	00:00:01
3	INDEX UNIQUE SCAN	PK_ORGANIZATION	1		1 (0)	00:00:01

可以看到 SQL 走的是 INDEX UNIQUE SCAN，说明表上不缺索引，设计上没有问题。这里简单点看历史执行计划

也可以这样：

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_AWR(SQL_ID =>
'1cmnjddakrqbv' )) ;
```

```
SQL_ID 1cmnjddakrqbv
--------------------
update organization o set o.quota_unused = o.quota_unused-:1,o.mod_date
= systimestamp where o.quota_unused >= :2 and o.bank_id=:3 and
o.pro_id=:4

Plan hash value: 374610298

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT             |                 |       |       |     2 (100)|          |
|   1 |  UPDATE                      | ORGANIZATION    |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| ORGANIZATION    |     1 |    30 |     2   (0)| 00:00:01 |
|   3 |    INDEX UNIQUE SCAN         | PK_ORGANIZATION |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

我们继续查看该 SQL 的绑定变量的值具体是多少：

```
SELECT * FROM V$SQL_BIND_CAPTURE  A WHERE A.SQL_ID IN ('1cmnjddakrqbv') ;
SELECT A.SQL_ID,A.NAME,A.POSITION,A.DATATYPE_STRING,A.VALUE_STRING FROM
V$SQL_BIND_CAPTURE  A WHERE A.SQL_ID IN ('1cmnjddakrqbv') ;
```

	SQL_ID	NAME	POSITION	DATATYPE_STRING	VALUE_STRING
1	1cmnjddakrqbv	:1	1	NUMBER	
2	1cmnjddakrqbv	:2	2	NUMBER	1
3	1cmnjddakrqbv	:3	3	VARCHAR2(32)	17856
4	1cmnjddakrqbv	:4	4	VARCHAR2(128)	HSB201602

这里准确点我们应该查询 DBA_HIST_SQLBIND 该视图，如下：

```
SELECT   A.sql_id,A.name,A.datatype_string,A.value_string,COUNT(1)
    FROM DBA_HIST_SQLBIND A
   WHERE A.SQL_ID IN ('1cmnjddakrqbv')
     AND A.SNAP_ID BETWEEN 1145 AND 1148
       GROUP BY A.sql_id,A.name,A.datatype_string,A.value_string
     ;
```

	SQL_ID	NAME	DATATYPE_STRING	VALUE_STRING	COUNT(1)
2	1cmnjddakrqbv	:1	NUMBER		6
1	1cmnjddakrqbv	:2	NUMBER	1	6
4	1cmnjddakrqbv	:3	VARCHAR2(32)	17856	2
3	1cmnjddakrqbv	:3	VARCHAR2(32)	05612	4
5	1cmnjddakrqbv	:4	VARCHAR2(128)	HSB201602	6

结果差不多。由此找到了产生等待的 SQL 语句，我们查询一下：

```
SELECT * FROM CNSL.ORGANIZATION o WHERE O.QUOTA_UNUSED >= 1
   AND O.BANK_ID IN ( '17856'  , '05612' )
   AND O.PRO_ID = 'HSB201602';
```

ORG_ID	BANK_ID	BANK_NAME	PROVINCE	CITY	COUNTY	ADDRESS	M
3005	05612	[redacted]发区支行	黑龙江省	黑河市	测试县	[redacted]街133号	04
2627	17856	[redacted]步支行	广东省	深圳市	测试县	[redacted]航都大厦一楼	07

查询出来也就 2 行的数据，说明整个过程中，500 多的会话就在更新这 2 条记录，那肯定会产生行锁等待的。

1.3.4.1 ADDM 的建议

最后突然想到了 ADDM，于是做了个 ADDM 查询。

```
DECLARE
  TASK_NAME VARCHAR2(50) := 'HEALTH_CHECK_BY_LHR';
  TASK_DESC VARCHAR2(50) := 'HEALTH_CHECK_BY_LHR';
  TASK_ID   NUMBER;
BEGIN
  DBMS_ADVISOR.CREATE_TASK('ADDM', TASK_ID, TASK_NAME, TASK_DESC, NULL);
  DBMS_ADVISOR.SET_TASK_PARAMETER(TASK_NAME, 'START_SNAPSHOT', 1145);
  DBMS_ADVISOR.SET_TASK_PARAMETER(TASK_NAME, 'END_SNAPSHOT', 1148);
  DBMS_ADVISOR.SET_TASK_PARAMETER(TASK_NAME, 'INSTANCE', 1);
  DBMS_ADVISOR.SET_TASK_PARAMETER(TASK_NAME, 'DB_ID', 3860591551);
  DBMS_ADVISOR.EXECUTE_TASK(TASK_NAME);
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
```

执行完成后 ADDM 报告就创建好了，然后我们用函数 DBMS_ADVISOR.GET_TASK_REPORT 来获取报告：

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT('HEALTH_CHECK_BY_LHR', 'TEXT', 'ALL')
ADDM_RESULTS
FROM DUAL;
```

	ADDM_RESULTS
1	<CLOB> ...

查看 ADDM 具体内容，重要的内容我用红色标注出来了：

```
        ADDM Report for Task 'HEALTH_CHECK_BY_LHR'
        ------------------------------------------

Analysis Period
---------------
AWR snapshot range from 1145 to 1148.
Time period starts at 31-AUG-16 05.33.52 PM
Time period ends at 31-AUG-16 06.14.36 PM

Analysis Target
---------------
Database 'ORACNSL' with DB ID 3860591551.
Database version 11.2.0.4.0.
ADDM performed an analysis of instance oraCNSL1, numbered 1 and hosted at
ZFCNSLDB1.

Activity During the Analysis Period
-----------------------------------
Total database time was 910831 seconds.
The average number of active sessions was 372.68.
```

```
Summary of Findings
-------------------
   Description               Active Sessions      Recommendations
                             Percent of Activity
   ------------------------  ------------------   --------------
1  Top SQL Statements        371.83 | 99.77       1
2  Row Lock Waits            350.15 | 93.96       1
3  Buffer Busy - Hot Objects 20.62 | 5.53         1
4  Buffer Busy - Hot Block   20.6 | 5.53          2
5  "Cluster" Wait Class      17.7 | 4.75          0
6  Global Cache Messaging    5.86 | 1.57          1


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


         Findings and Recommendations
         ----------------------------

Finding 1: Top SQL Statements
Impact is 371.83 active sessions, 99.77% of total activity.
-----------------------------------------------------------
SQL statements consuming significant database time were found. These
statements offer a good opportunity for performance improvement.

   Recommendation 1: SQL Tuning
   Estimated benefit is 371.83 active sessions, 99.77% of total activity.
   ------------------------------------------------------------------------
   Action
      Investigate the UPDATE statement with SQL_ID "1cmnjddakrqbv" for
      possible performance improvements. You can supplement the information
      given here with an ASH report for this SQL_ID.
      Related Object
         SQL statement with SQL_ID 1cmnjddakrqbv.
         update organization o set o.quota_unused =
         o.quota_unused-:1,o.mod_date = systimestamp where o.quota_unused >=
         :2 and o.bank_id=:3 and o.pro_id=:4
   Rationale
      The SQL spent only 4% of its database time on CPU, I/O and Cluster
      waits. Therefore, the SQL Tuning Advisor is not applicable in this case.
      Look at performance data for the SQL to find potential improvements.
   Rationale
      Database time for this SQL was divided as follows: 100% for SQL
      execution, 0% for parsing, 0% for PL/SQL execution and 0% for Java
      execution.
   Rationale
      SQL statement with SQL_ID "1cmnjddakrqbv" was executed 377427 times and
      had an average elapsed time of 2.4 seconds.
   Rationale
      Waiting for event "enq: TX - row lock contention" in wait class
      "Application" accounted for 94% of the database time spent in processing
      the SQL statement with SQL_ID "1cmnjddakrqbv".


Finding 2: Row Lock Waits
Impact is 350.15 active sessions, 93.96% of total activity.
-----------------------------------------------------------
SQL statements were found waiting for row lock waits.

   Recommendation 1: Application Analysis
   Estimated benefit is 350.15 active sessions, 93.96% of total activity.
   ------------------------------------------------------------------------
   Action
      Significant row contention was detected in the TABLE "CNSL.ORGANIZATION"
      with object ID 87620. Trace the cause of row contention in the
      application logic using the given blocked SQL.
      Related Object
         Database object with ID 87620.
   Rationale
      The SQL statement with SQL_ID "1cmnjddakrqbv" was blocked on row locks.
      Related Object
         SQL statement with SQL_ID 1cmnjddakrqbv.
         update organization o set o.quota_unused =
```

```
        o.quota_unused-:1,o.mod_date = systimestamp where o.quota_unused >=
        :2 and o.bank_id=:3 and o.pro_id=:4

   Symptoms That Led to the Finding:
   ---------------------------------
      Wait class "Application" was consuming significant database time.
      Impact is 350.15 active sessions, 93.96% of total activity.


Finding 3: Buffer Busy - Hot Objects
Impact is 20.62 active sessions, 5.53% of total activity.
-----------------------------------------------------------
Read and write contention on database blocks was consuming significant
database time.

   Recommendation 1: Schema Changes
   Estimated benefit is 20.6 active sessions, 5.53% of total activity.
   -------------------------------------------------------------------
   Action
      Consider rebuilding the TABLE "CNSL.ORGANIZATION" with object ID 87620
      using a higher value for PCTFREE.
      Related Object
         Database object with ID 87620.

   Symptoms That Led to the Finding:
   ---------------------------------
      Read and write contention on database blocks was consuming significant
      database time.
      Impact is 20.62 active sessions, 5.53% of total activity.
         Inter-instance messaging was consuming significant database time on
         this instance.
         Impact is 5.86 active sessions, 1.57% of total activity.
            Wait class "Cluster" was consuming significant database time.
            Impact is 17.7 active sessions, 4.75% of total activity.


Finding 4: Buffer Busy - Hot Block
Impact is 20.6 active sessions, 5.53% of total activity.
-----------------------------------------------------------
A hot data block with concurrent read and write activity was found. The block
belongs to segment "CNSL.ORGANIZATION" and is block 171 in file 14.

   Recommendation 1: Application Analysis
   Estimated benefit is 20.6 active sessions, 5.53% of total activity.
   -------------------------------------------------------------------
   Action
      Investigate application logic to find the cause of high concurrent read
      and write activity to the data present in this block.
      Related Object
         Database block with object number 87620, file number 14 and block
         number 171.

   Recommendation 2: Schema Changes
   Estimated benefit is 20.6 active sessions, 5.53% of total activity.
   -------------------------------------------------------------------
   Action
      Consider rebuilding the TABLE "CNSL.ORGANIZATION" with object ID 87620
      using a higher value for PCTFREE.
      Related Object
         Database object with ID 87620.

   Symptoms That Led to the Finding:
   ---------------------------------
      Read and write contention on database blocks was consuming significant
      database time.
      Impact is 20.62 active sessions, 5.53% of total activity.
         Inter-instance messaging was consuming significant database time on
         this instance.
         Impact is 5.86 active sessions, 1.57% of total activity.
            Wait class "Cluster" was consuming significant database time.
            Impact is 17.7 active sessions, 4.75% of total activity.


Finding 5: "Cluster" Wait Class
Impact is 17.7 active sessions, 4.75% of total activity.
```

```
---------------------------------------------------------
Wait class "Cluster" was consuming significant database time.

   No recommendations are available.


Finding 6: Global Cache Messaging
Impact is 5.86 active sessions, 1.57% of total activity.
---------------------------------------------------------
Inter-instance messaging was consuming significant database time on this
instance.

   Recommendation 1: Application Analysis
   Estimated benefit is 5.86 active sessions, 1.57% of total activity.
   ------------------------------------------------------------------
   Action
      Look at the "Top SQL Statements" finding for SQL statements consuming
      significant time on Cluster waits. For example, the UPDATE statement
      with SQL_ID "1cmnjddakrqbv" is responsible for 100% of Cluster wait
      during the analysis period.

   Symptoms That Led to the Finding:
   ---------------------------------
      Wait class "Cluster" was consuming significant database time.
      Impact is 17.7 active sessions, 4.75% of total activity.



~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        Additional Information
        ----------------------

Miscellaneous Information
-------------------------
Wait class "Commit" was not consuming significant database time.
Wait class "Concurrency" was not consuming significant database time.
Wait class "Configuration" was not consuming significant database time.
CPU was not a bottleneck for the instance.
Wait class "Network" was not consuming significant database time.
Wait class "User I/O" was not consuming significant database time.
The network latency of the cluster interconnect was within acceptable limits
of 1 milliseconds.
Session connect and disconnect calls were not consuming significant database
time.
Hard parsing of SQL statements was not consuming significant database time.
```

看来 ADDM 可以**一针见血**的找到系统存在的问题。

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-2124369/ 博客园地址：http://www.cnblogs.com/lhrbest/articles/5831147.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 2016-09-01 09:00~2016-09-01 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。