【故障处理】队列等待之 TX - allocate ITL entry 引起的死锁处理

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 - allocate ITL entry 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 ITL 死锁解决

⑤ ITL 死锁模拟

⑥ Merge 语句的非关联形式的查询优化

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 若网页文章代码格式有错乱，推荐使用 360 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，也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_XLHRD_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	11.2.0.3.0
db 存储	ASM
OS 版本及 kernel 版本	AIX 64 位 7.1.0.0

1.3.2 故障发生现象及故障分析解决

早上刚来上班，同事就发了一个 SQL 过来，说是有锁，然后我就查了查系统里的锁，结果一个锁都没得。好吧，还是得干点事的，先看看 SQL 语句：

```
MERGE INTO TLHR.TLHRBOKBAL S
USING (SELECT A.BOOKACCOUNT AS BOOKACCOUNT,
              (A.CURRBALANCE + NVL(B.BAL, 0.00)) AS BANKAMT
         FROM TLHR.TLHRBOKBAL_TMP A,
              (SELECT T1.BOOKACCOUNT AS BOOKACCOUNT,
                      SUM(DECODE(T1.DCFLAG, 'D', -T1.AMT, 'C', T1.AMT, 0)) AS BAL
                 FROM TLHR.TLHRBOKBALJN T1
                WHERE T1.BOOKACCOUNT LIKE '13500000%'
                  AND T1.TRANDATE = '20150901'
                  AND (T1.REASON IN ('1', '2') OR
                      (T1.REASON = '0' AND T1.ONLINEFLAG = '1'))
                GROUP BY T1.BOOKACCOUNT) B
        WHERE A.BOOKACCOUNT = B.BOOKACCOUNT(+)
          AND A.BOOKACCOUNT LIKE '13500000%') T
ON (S.BOOKACCOUNT = T.BOOKACCOUNT)
WHEN MATCHED THEN
  UPDATE
     SET S.LASTBALANCE = T.BANKAMT,
         S.CURRBALANCE = T.BANKAMT,
         S.DEBITAMT    = 0.00,
         S.CREDITAMT   = 0.00;
```

看起来是一个 MERGE 语句，按照小麦苗以前的经验，这一类的 SQL 最好是修改为 MERGE 的非关联形式比较好，我们先看看执行计划有没有问题：

先找到 SQL_ID 为 53qv858pwwwwb：

```
SELECT a.ELAPSED_TIME,a.EXECUTIONS,a.* FROM v$sql a WHERE a.SQL_TEXT LIKE
'%MERGE INTO TLHRBOKBAL S%' AND A.SQL_TEXT LIKE '%13500000%'  ;
```

查询历史执行计划：

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_AWR(SQL_ID => '53qv858pwwwwb' )) ;
```

```
Plan hash value: 2695089823

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                       | Name             | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT                 |                  |       |       |       |  155K(100)|          |
|   1 |  MERGE                          | TLHRBOKBAL       |       |       |       |           |          |
|   2 |   VIEW                          |                  |       |       |       |           |          |
|   3 |    HASH JOIN RIGHT OUTER        |                  |  153K|   15M|       |  155K   (2)| 00:31:04 |
|   4 |     VIEW                        |                  |     1 |    31 |       |     6   (0)| 00:00:01 |
|   5 |      SORT GROUP BY              |                  |     1 |    41 |       |     6   (0)| 00:00:01 |
|   6 |       TABLE ACCESS BY INDEX ROWID| TLHRBOKBALJN     |     1 |    41 |       |     6   (0)| 00:00:01 |
|   7 |        INDEX RANGE SCAN         | PK_TLHRBOKBALJN  |     2 |       |       |     4   (0)| 00:00:01 |
|   8 |     HASH JOIN                   |                  |  153K|   10M|  5848K|  155K   (2)| 00:31:04 |
|   9 |      TABLE ACCESS FULL          | TLHRBOKBAL_TMP   |  153K| 4048K|       | 85415   (2)| 00:17:05 |
|  10 |      TABLE ACCESS FULL          | TLHRBOKBAL       |  305K|   13M|       | 68755   (3)| 00:13:46 |
-----------------------------------------------------------------------------------------------------------
```

可以看到，该执行计划的顺序为【7-->6-->5-->4-->9-->10-->8-->3-->2-->1-->0】，而耗费性能的地方在 9、10、8 这 3 个步骤上，走的是全表扫描，我们先看看 2 个大表的数据量：

```
    SELECT COUNT(*)  FROM TLHR.TLHRBOKBAL_TMP A WHERE  A.BOOKACCOUNT LIKE
'13500000%';   --306043/38998765
    SELECT COUNT(*) FROM TLHR.TLHRBOKBAL A WHERE  A.BOOKACCOUNT LIKE
'13500000%';    --306043/38826275
```

从 3000 万的数据里边取出 30 万的数据，还是比较少的，所以应该去走索引的，看了一下统计信息，也是最新收集的，好吧，算了，先修改一下 SQL 让其走索引扫描看看,：

```
    MERGE  INTO TLHR.TLHRBOKBAL S
    USING (SELECT S.ROWID ROWIDS,
                 A.BOOKACCOUNT AS BOOKACCOUNT,
                 (A.CURRBALANCE + NVL(B.BAL, 0.00)) AS BANKAMT
            FROM (SELECT /*+index(NB,PK_TLHRBOKBAL_TMP)*/NB.CURRBALANCE,NB.BOOKACCOUNT
                   FROM TLHR.TLHRBOKBAL_TMP NB
                  WHERE NB.BOOKACCOUNT LIKE '13500000%') A,
                 TLHR.TLHRBOKBAL S,
                 (SELECT T1.BOOKACCOUNT AS BOOKACCOUNT,
                        SUM(DECODE(T1.DCFLAG, 'D', -T1.AMT, 'C', T1.AMT, 0)) AS BAL
                   FROM TLHR.TLHRBOKBALJN T1
                  WHERE T1.BOOKACCOUNT LIKE '13500000%'
                    AND T1.TRANDATE = '20150901'
                    AND (T1.REASON IN ('1', '2') OR
                        (T1.REASON = '0' AND T1.ONLINEFLAG = '1'))
                  GROUP BY T1.BOOKACCOUNT) B
           WHERE A.BOOKACCOUNT = B.BOOKACCOUNT(+)
             AND S.BOOKACCOUNT = A.BOOKACCOUNT
             AND S.BOOKACCOUNT LIKE '13500000%') T
    ON (T.ROWIDS = S.ROWID)
    WHEN MATCHED THEN
      UPDATE
         SET S.LASTBALANCE = T.BANKAMT,
            S.CURRBALANCE = T.BANKAMT,
            S.DEBITAMT    = 0.00,
            S.CREDITAMT   = 0.00
```

```
 Plan Hash Value  : 273017430

------------------------------------------------------------------------------------------------------------
| Id  | Operation                        | Name              | Rows   | Bytes    | Cost   | Time     |
------------------------------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT                  |                   | 152885 |  4280780 | 283362 | 00:56:41 |
|   1 |  MERGE                           | TLHRBOKBAL        |        |          |        |          |
|   2 |   VIEW                           |                   |        |          |        |          |
|   3 |    NESTED LOOPS                  |                   | 152885 | 20945245 | 283362 | 00:56:41 |
| * 4 |     HASH JOIN RIGHT OUTER        |                   | 152885 | 14065420 | 130342 | 00:26:05 |
|   5 |      VIEW                        |                   |    124 |     3844 |  15668 | 00:03:09 |
|   6 |       SORT GROUP BY              |                   |    124 |     5084 |  15668 | 00:03:09 |
| * 7 |        TABLE ACCESS BY INDEX ROWID | TLHRBOKBALJN    |    124 |     5084 |  15668 | 00:03:09 |
| * 8 |         INDEX RANGE SCAN         | PK_TLHRBOKBALJN   |    165 |          |  15501 | 00:03:07 |
| * 9 |      HASH JOIN                   |                   | 152885 |  9325985 | 114671 | 00:22:57 |
|  10 |       TABLE ACCESS BY INDEX ROWID | TLHRBOKBAL_TMP   | 153563 |  4146201 | 112930 | 00:22:36 |
| * 11 |        INDEX RANGE SCAN          | PK_TLHRBOKBAL_TMP | 153563 |          |   1159 | 00:00:14 |
| * 12 |       INDEX RANGE SCAN           | PK_TLHRBOKBAL     | 152884 |  5198056 |   1117 | 00:00:14 |
|  13 |     TABLE ACCESS BY USER ROWID   | TLHRBOKBAL        |      1 |       45 |      1 | 00:00:01 |
------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
------------------------------------------
* 4 - access("NB"."BOOKACCOUNT"="B"."BOOKACCOUNT"(+))
* 7 - filter("T1"."REASON"='0' AND "T1"."ONLINEFLAG"='1' OR "T1"."REASON"='1' OR "T1"."REASON"='2')
* 8 - access("T1"."BOOKACCOUNT" LIKE '13500000%' AND "T1"."TRANDATE"='20150901')
* 8 - filter("T1"."BOOKACCOUNT" LIKE '13500000%' AND "T1"."TRANDATE"='20150901')
* 9 - access("S"."BOOKACCOUNT"="NB"."BOOKACCOUNT")
* 11 - access("NB"."BOOKACCOUNT" LIKE '13500000%')
* 11 - filter("NB"."BOOKACCOUNT" LIKE '13500000%')
* 12 - access("S"."BOOKACCOUNT" LIKE '13500000%')
* 12 - filter("S"."BOOKACCOUNT" LIKE '13500000%')
```

执行计划中，基本都走了索引了，跑了一下，大约 1 分种多，但是里边有个 HINTS，分析了一下表 TLHRBOKBAL_TMP 上的索引情况，发现是个主键索引，且有 2 个列（BOOKACCOUNT,CURRENCY），但是不包含列 CURRBALANCE，可能是 Oracle 觉得回表读的耗费比较大吧，那这里可以使用虚拟索引测试一下索引的性能：

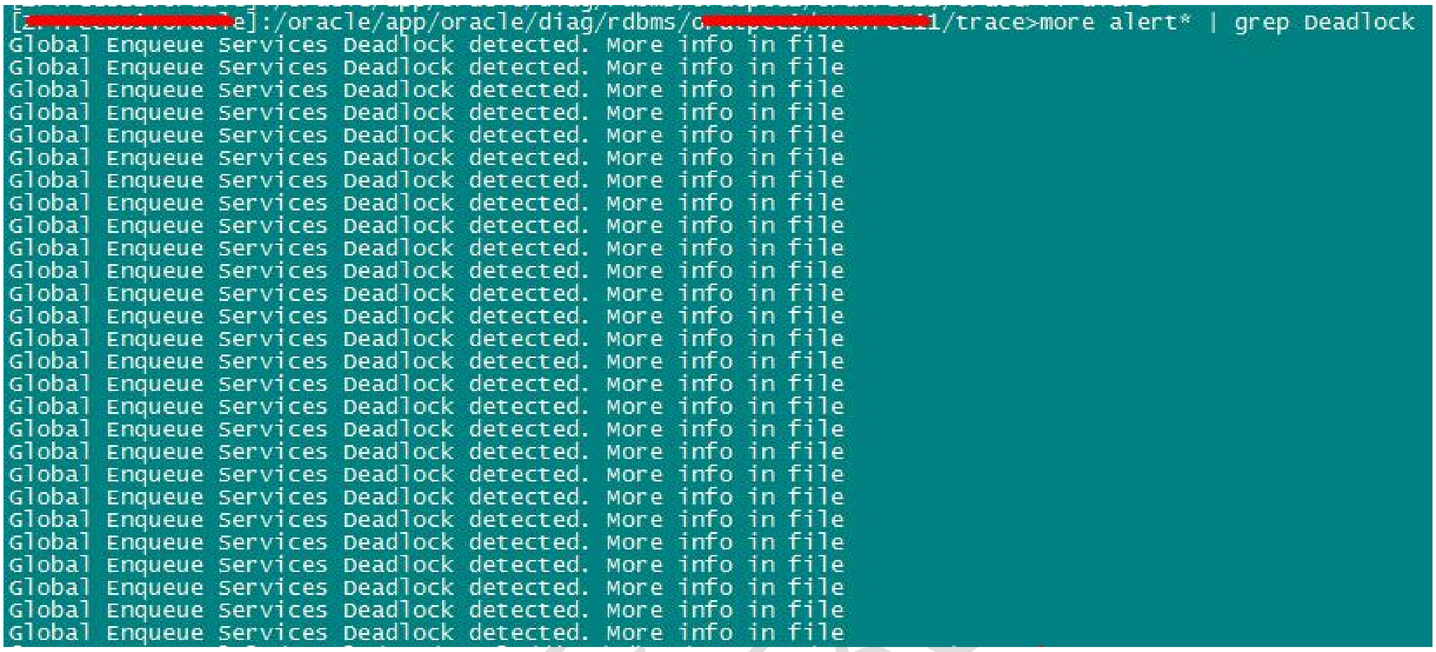
```
CREATE INDEX IX_VI01_ID ON TLHR.TLHRBOKBAL_TMP(CURRBALANCE, CURRENCY,BOOKACCOUNT)
NOSEGMENT;
ALTER SESSION SET "_USE_NOSEGMENT_INDEXES"=TRUE;
    EXPLAIN PLAN FOR  MERGE INTO TLHR.TLHRBOKBAL S
    USING (SELECT S.ROWID ROWIDS,
                  A.BOOKACCOUNT AS BOOKACCOUNT,
                  (A.CURRBALANCE + NVL(B.BAL, 0.00)) AS BANKAMT
             FROM (SELECT NB.CURRBALANCE,NB.BOOKACCOUNT
                     FROM TLHR.TLHRBOKBAL_TMP NB
                    WHERE NB.BOOKACCOUNT LIKE '13500000%') A,
                  TLHR.TLHRBOKBAL S,
                (SELECT T1.BOOKACCOUNT AS BOOKACCOUNT,
                        SUM(DECODE(T1.DCFLAG, 'D', -T1.AMT, 'C', T1.AMT, 0)) AS BAL
                   FROM TLHR.TLHRBOKBALJN T1
                  WHERE T1.BOOKACCOUNT LIKE '13500000%'
                    AND T1.TRANDATE = '20150901'
                    AND (T1.REASON IN ('1', '2') OR
                        (T1.REASON = '0' AND T1.ONLINEFLAG = '1'))
                  GROUP BY T1.BOOKACCOUNT) B
          WHERE A.BOOKACCOUNT = B.BOOKACCOUNT(+)
            AND S.BOOKACCOUNT = A.BOOKACCOUNT
            AND S.BOOKACCOUNT LIKE '13500000%') T
    ON (T.ROWIDS = S.ROWID)
    WHEN MATCHED THEN
      UPDATE
         SET S.LASTBALANCE = T.BANKAMT,
             S.CURRBALANCE = T.BANKAMT,
             S.DEBITAMT    = 0.00,
         S.CREDITAMT    = 0.00;
SELECT * FROM TABLE(DBMS_XPLAN.display);
```

```
Plan hash value: 983878991

---------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name            | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT                |                 |  152K|  4180K|       |   170K  (1)| 00:34:06 |
|   1 |  MERGE                         | TLHRBOKBAL      |      |       |       |            |          |
|   2 |   VIEW                         |                 |      |       |       |            |          |
|   3 |    NESTED LOOPS                |                 |  152K|    19M|       |   170K  (1)| 00:34:06 |
|*  4 |     HASH JOIN RIGHT OUTER      |                 |  152K|    13M|       | 17421   (1)| 00:03:30 |
|   5 |      VIEW                      |                 |  124 |  3844 |       | 15668   (1)| 00:03:09 |
|   6 |       SORT GROUP BY            |                 |  124 |  5084 |       | 15668   (1)| 00:03:09 |
|*  7 |        TABLE ACCESS BY INDEX ROWID| TLHRBOKBALJN |  124 |  5084 |       | 15668   (1)| 00:03:09 |
|*  8 |         INDEX RANGE SCAN       | PK_TLHRBOKBALJN |  165 |       |       | 15501   (1)| 00:03:07 |
|*  9 |      HASH JOIN                 |                 |  152K|  9107K|  5856K|  1750   (1)| 00:00:22 |
|* 10 |       INDEX FAST FULL SCAN     | IX_VI01_ID      |  153K|  4049K|       |     9   (0)| 00:00:01 |
|* 11 |       INDEX RANGE SCAN         | PK_TLHRBOKBAL   |  152K|  5076K|       |  1117   (1)| 00:00:14 |
|  12 |     TABLE ACCESS BY USER ROWID | TLHRBOKBAL      |     1 |    45 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("NB"."BOOKACCOUNT"="B"."BOOKACCOUNT"(+))
   7 - filter("T1"."REASON"='0' AND "T1"."ONLINEFLAG"='1' OR "T1"."REASON"='1' OR "T1"."REASON"='2')
   8 - access("T1"."BOOKACCOUNT" LIKE '13500000%' AND "T1"."TRANDATE"='20150901')
       filter("T1"."BOOKACCOUNT" LIKE '13500000%' AND "T1"."TRANDATE"='20150901')
   9 - access("S"."BOOKACCOUNT"="NB"."BOOKACCOUNT")
  10 - filter("NB"."BOOKACCOUNT" LIKE '13500000%')
  11 - access("S"."BOOKACCOUNT" LIKE '13500000%')
       filter("S"."BOOKACCOUNT" LIKE '13500000%')
```

说明创建 3 个列的索引是可以的。我们先将该虚拟索引删除 DROP INDEX IX_VI01_ID;

1.3.2.1 ITL 死锁问题解决

另外一个问题，是开发说上边的 SQL 语句产生了死锁，起初我还半信半疑，先去告警日志中用命令(more alert* | grep Deadlock) 搜了一下：



```
[[redacted]e]:/oracle/app/oracle/diag/rdbms/[redacted]1/trace>more alert* | grep Deadlock
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
Global Enqueue Services Deadlock detected. More info in file
```

结果发现很多的死锁，拿到相关的文件，看到如下一段：

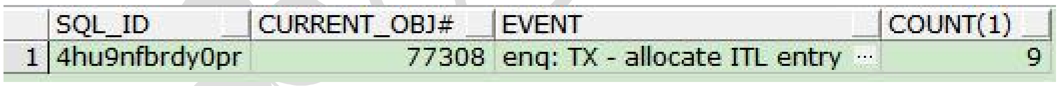
```
user session for deadlock lock 0x7000008094d14e0
  sid: 332 ser: 47221 audsid: 991000 user: 84/TLHR
    flags: (0x41) USR/- flags_idl: (0x1) BSY/-/-/-/-/-
    flags2: (0x40009) -/-/INC
  pid: 101 O/S info: user: grid, term: UNKNOWN, ospid: 6489034
    image: oracle@ZFTLHRDB1
  client details:
    O/S info: user: TLHR, term: , ospid: 34406578
    machine: ZFTLHRAP1 program: bat_CheckBookBal@ZFTLHRAP1 (TNS V1-V3)
    application name: bat_CheckBookBal@ZFTLHRAP1 (TNS V1-V3), hash value=446537749
  current SQL:
   MERGE INTO TLHRBOKBAL S USING  (SELECT A.BOOKACCOUNT AS BOOKACCOUNT,  (A.CURRBALANCE + nvl(B.BAL, 0.00)) AS BANKAMT  FROM  TLHRBOKBAL_TMP A, (SELECT T1.BOOKACCOUNT AS BOOKACCOUNT,  SUM(DECODE(T1.DCFLAG, 'D', -T1.AMT, 'C', T1.AMT, 0)) AS BAL  FROM TLHRBOKBALJN T1  WHERE T1.BOOKACCOUNT LIKE '13450000'||'%' AND T1.TRANDATE='20160901'  AND (T1.REASON = '2' OR T1.REASON = '1' OR  (T1.REASON = '0' AND T1.ONLINEFLAG = '1'))  GROUP BY T1.BOOKACCOUNT) B  WHERE A.BOOKACCOUNT = B.BOOKACCOUNT(+)  AND A.BOOKACCOUNT LIKE '13450000'||'%') T  ON (S.BOOKACCOUNT = T.BOOKACCOUNT)  WHEN MATCHED THEN UPDATE  SET S.LASTBALANCE=T.BANKAMT,S.CURRBALANCE=T.BANKAMT,S.DEBITAMT=0.00,S.CREDITAMT=0.00
DUMP LOCAL BLOCKER: initiate state dump for DEADLOCK
  possible owner[101.6489034] on resource TX-00EE0009-00005EA6
*** 2016-09-01 18:30:38.014
Submitting asynchronized dump request [28]. summary=[ges process stack dump (kjdglblkrdm1)].
Global blockers dump end:-----------------------------------
Global Wait-For-Graph(WFG) at ddTS[0.2fe0] :
BLOCKED 0x7000008e9c8bc28 3 wq 2 cvtops x1 TX 0x159001e.0x2379(ext 0x5,0x0)[1006-0065-0019365C] inst 1
BLOCKER 0x700000809ab4b28 3 wq 1 cvtops x28 TX 0x159001e.0x2379(ext 0x5,0x0)[2005-005E-00185E15] inst 2
BLOCKED 0x700000891b48708 3 wq 2 cvtops x1 TX 0x1c2001d.0x4b82(ext 0x2,0x0)[2005-005E-00185E15] inst 2
BLOCKER 0x7000008e9c8b148 3 wq 1 cvtops x28 TX 0x1c2001d.0x4b82(ext 0x2,0x0)[1004-004D-0000C03E] inst 1
```

```
BLOCKED 0x70000089a636970 3 wq 2 cvtops x1 TX 0x1c0000b.0x18f6(ext 0x2,0x0)[1004-004D-0000C03E] inst 1
BLOCKER 0x7000008e9c8b4e8 3 wq 1 cvtops x28 TX 0x1c0000b.0x18f6(ext 0x2,0x0)[1005-0058-000DD3D9] inst 1
BLOCKED 0x700000891d5fc50 3 wq 2 cvtops x1 TX 0xee0009.0x5ea6(ext 0x2,0x0)[1005-0058-000DD3D9] inst 1
BLOCKER 0x7000008094d14e0 3 wq 1 cvtops x28 TX 0xee0009.0x5ea6(ext 0x2,0x0)[1006-0065-0019365C] inst 1
* Cancel deadlock victim lockp 0x7000008e9c8bc28
*** 2016-09-01 18:30:43.001
kjddt2vb: valblk  [0.2fe1] > local ts [0.2fe0]
*** 2016-09-01 18:30:47.000
kjddt2vb: valblk  [0.2fe5] > local ts [0.2fe4]
*** 2016-09-01 18:40:38.062
kjddt2vb: valblk  [0.2ff1] > local ts [0.2ff0]
*** 2016-09-01 18:42:01.084
kjddt2vb: valblk  [0.2ff4] > local ts [0.2ff3]
2016-09-01 22:33:52.213848 : Setting 3-way CR grants to 0 global-lru off? 0
*** 2016-09-01 22:34:23.163
2016-09-01 22:34:23.163681 : Setting 3-way CR grants to 1 global-lru off? 0
2016-09-01 22:50:00.603305 : Setting 3-way CR grants to 0 global-lru off? 0
*** 2016-09-01 22:51:33.104
2016-09-01 22:51:33.104615 : Setting 3-way CR grants to 1 global-lru off? 0
2016-09-02 05:30:18.751891 : Setting 3-way CR grants to 0 global-lru off? 0
2016-09-02 05:49:01.360730 : Setting 3-way CR grants to 1 global-lru off? 0
2016-09-02 10:28:55.429293 : Setting 3-way CR grants to 0 global-lru off? 0
```

果然，产生死锁的 SQL 还是上边分析优化的 SQL,其中会话信息为：（332，47221），我们去

DBA_HIST_ACTIVE_SESS_HISTORY 视图里查询：

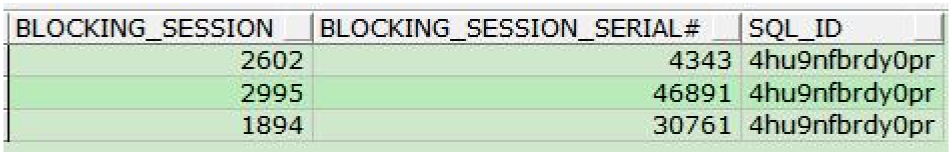
```
SELECT D.SQL_ID, D.CURRENT_OBJ#, D.EVENT, COUNT(1)
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN
       TO_DATE('2016-09-01 18:25:00', 'YYYY-MM-DD HH24:MI:SS') AND
       TO_DATE('2016-09-01 18:45:00', 'YYYY-MM-DD HH24:MI:SS')
   AND D.BLOCKING_SESSION_STATUS = 'VALID'
   AND D.SESSION_ID = 332
   AND D.SESSION_SERIAL# = 47221
 GROUP BY D.SQL_ID, D.CURRENT_OBJ#, D.EVENT;
```



	SQL_ID	CURRENT_OBJ#	EVENT	COUNT(1)
1	4hu9nfbrdy0pr	77308	enq: TX - allocate ITL entry ···	9

可以看到该会话的等待事件是 enq: TX - allocate ITL entry。可以猜测是由于 ITL 事务槽引起的问题。

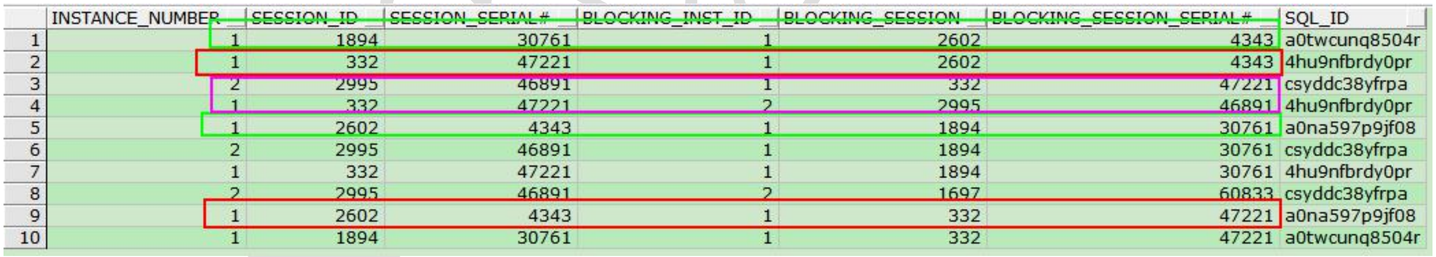
```
SELECT DISTINCT D.BLOCKING_SESSION, D.BLOCKING_SESSION_SERIAL#, D.SQL_ID
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN
       TO_DATE('2016-09-01 18:25:00', 'YYYY-MM-DD HH24:MI:SS') AND
       TO_DATE('2016-09-01 18:45:00', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: TX - allocate ITL entry'
   AND D.BLOCKING_SESSION_STATUS = 'VALID'
   AND D.SESSION_ID = 332
   AND D.SESSION_SERIAL# = 47221;
```



BLOCKING_SESSION	BLOCKING_SESSION_SERIAL#	SQL_ID
2602	4343	4hu9nfbrdy0pr
2995	46891	4hu9nfbrdy0pr
1894	30761	4hu9nfbrdy0pr

可以看出会话（332，47221）共阻塞了 3 个会话，由于有死锁，那么我们看看上边查询出来的 3 个会话阻塞了哪些会话：

```
SELECT DISTINCT D.INSTANCE_NUMBER,
                D.SESSION_ID,
                D.SESSION_SERIAL#,
                D.BLOCKING_INST_ID,
                D.BLOCKING_SESSION,
                D.BLOCKING_SESSION_SERIAL#,
                D.SQL_ID
  FROM DBA_HIST_ACTIVE_SESS_HISTORY D
 WHERE D.SAMPLE_TIME BETWEEN
       TO_DATE('2016-09-01 18:25:00', 'YYYY-MM-DD HH24:MI:SS') AND
       TO_DATE('2016-09-01 18:45:00', 'YYYY-MM-DD HH24:MI:SS')
   AND D.EVENT = 'enq: TX - allocate ITL entry'
   AND D.BLOCKING_SESSION_STATUS = 'VALID'
   AND ((D.SESSION_ID = 332 AND D.SESSION_SERIAL# = 47221) OR
       (D.SESSION_ID = 2602 AND D.SESSION_SERIAL# = 4343) OR
       (D.SESSION_ID = 2995 AND D.SESSION_SERIAL# = 46891) OR
       (D.SESSION_ID = 1894 AND D.SESSION_SERIAL# = 30761));
```

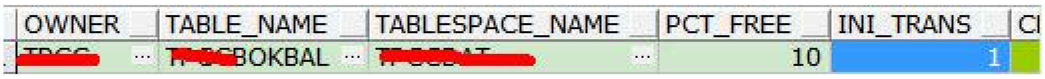


	INSTANCE_NUMBER	SESSION_ID	SESSION_SERIAL#	BLOCKING_INST_ID	BLOCKING_SESSION	BLOCKING_SESSION_SERIAL#	SQL_ID
1	1	1894	30761	1	2602	4343	a0twcunq8504r
2	1	332	47221	1	2602	4343	4hu9nfbrdy0pr
3	2	2995	46891	1	332	47221	csyddc38yfrpa
4	1	332	47221	2	2995	46891	4hu9nfbrdy0pr
5	1	2602	4343	1	1894	30761	a0na597p9jf08
6	2	2995	46891	1	1894	30761	csyddc38yfrpa
7	1	332	47221	1	1894	30761	4hu9nfbrdy0pr
8	2	2995	46891	2	1697	60833	csyddc38yfrpa
9	1	2602	4343	1	332	47221	a0na597p9jf08
10	1	1894	30761	1	332	47221	a0twcunq8504r

可以看到，1894 和 2602 相互阻塞（绿色表示），332 和 2602 相互阻塞（红色表示），2995 和 332 相互阻塞（粉色表示）这么多的相互阻塞就产生了死锁，这里由于 SQL_ID 不同，而且产生的等待事件是 enq: TX - allocate ITL entry，所以推测出生成的是 ITL 死锁。

解决这类问题就是增大 ini_trans 和 PCT_FREE 的值。

```
SELECT * FROM DBA_TABLES D WHERE D.TABLE_NAME = 'TLHRBOKBAL';
```



OWNER	TABLE_NAME	TABLESPACE_NAME	PCT_FREE	INI_TRANS	CI
[redacted] ...	T[redacted]BOKBAL ...	[redacted] ...	10	1	

可以看到，ini_trans 和 PCT_FREE 的值都是默认的，太小了，根据 MOS（**Troubleshooting waits for 'enq: TX -**

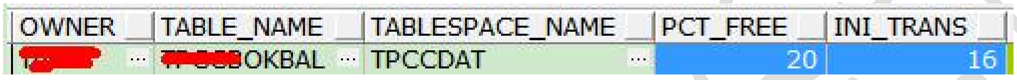
allocate ITL entry' (Doc ID 1472175.1) 地址：**http://blog.itpub.net/26736162/viewspace-2124531/**）我们需要修改该参数，SQL 如下：

```
ALTER TABLE TLHR.TLHRBOKBAL PCTFREE 20  INITRANS 16 ;
ALTER TABLE TLHR.TLHRBOKBAL MOVE NOLOGGING PARALLEL 12;
ALTER TABLE TLHR.TLHRBOKBAL LOGGING NOPARALLEL;
ALTER INDEX TLHR.PK_TLHRBOKBAL  REBUILD PCTFREE 20 INITRANS 16 NOLOGGING PARALLEL 12;
ALTER INDEX TLHR.PK_TLHRBOKBAL LOGGING NOPARALLEL;
```

由于表里有 3000W 的数据量，开了并行，本来我预估的是 5 分钟，结果 move 表的时候 10 秒都不到还是比较快的。

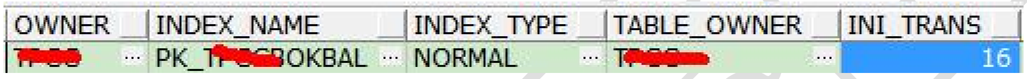
调整之后的值：

```
SELECT * FROM DBA_TABLES D WHERE D.TABLE_NAME = 'TLHRBOKBAL';
```



OWNER	TABLE_NAME	TABLESPACE_NAME	PCT_FREE	INI_TRANS
[redacted]	[redacted]OKBAL	TPCCDAT	20	16

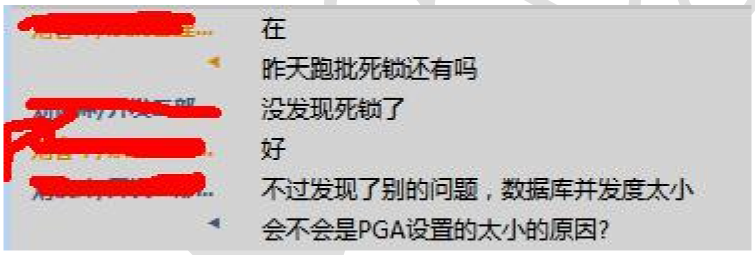
```
SELECT * FROM dba_indexes d WHERE d.index_name='PK_TLHRBOKBAL';
```



OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	INI_TRANS
[redacted]	PK_T[redacted]OKBAL	NORMAL	T[redacted]	16

修改已经生效，接下来就看开发那边是否还报死锁的错误，这个等待需要明天看了。

终于等到第 2 天了，看来没有报错了：



[redacted] 在
昨天跑批死锁还有吗
[redacted] 没发现死锁了
[redacted] 好
[redacted] 不过发现了别的问题，数据库并发度太小
会不会是PGA设置的太小的原因?

1.4 这里我们模拟一个 ITL 死锁

有人的地方就有江湖，有资源阻塞的地方就可能有死锁。所谓死锁： 是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。其最常见的死锁的类型分为：行级锁（row-level locks）和块级锁（block-level locks），这里的行级锁其实就是指的 ITL 死锁。有关死锁的问题，有许多需要介绍的，这篇 blog 主要是故障处理，所以这里我们模拟一个 ITL 死锁产生的过程即可，后边我会系统的发一次有关死锁的内

容，还有 ITL 的内容，希望大家持续关注小麦苗的微信公众号(xiaomaimiaolhr)。

实验部分：

实验的设计过程来源于网络！

我们首先创建一张表 T_ITL_LHR，这里指定 PCTFREE 为 0，INITRANS 为 1，就是为了观察到 ITL 的真实等待情况，然后我们给这些块内插入数据，把块填满，让它不能有空间分配。

```
SYS@lhrdb21> SELECT * FROM V$VERSION;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
PL/SQL Release 11.2.0.4.0 - Production
CORE    11.2.0.4.0      Production
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production
NLSRTL Version 11.2.0.4.0 - Production
SYS@lhrdb21> SHOW PARAMETER CLUSTER
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cluster_database                     boolean     TRUE
cluster_database_instances           integer     2
cluster_interconnects                string
SYS@lhrdb21> CREATE TABLE T_ITL_LHR(A INT) PCTFREE 0 INITRANS 1;
Table created.
SYS@lhrdb21> BEGIN
  2    FOR I IN 1 .. 2000 LOOP
  3      INSERT INTO T_ITL_LHR VALUES (I);
  4    END LOOP;
  5  END;
  6  /
PL/SQL procedure successfully completed.
SYS@lhrdb21> COMMIT;
Commit complete.
```

我们检查数据填充的情况：

```
SYS@lhrdb21> SELECT F, B, COUNT(*)
  2    FROM (SELECT DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID) F,
  3                 DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID) B
  4            FROM T_ITL_LHR)
  5   GROUP BY F, B
  6   ORDER BY F,B;
         F          B   COUNT(*)
---------- ---------- ----------
         1      94953        734
         1      94954        734
         1      94955        532
```

可以发现，这 2000 条数据分布在 3 个块内部，其中有 2 个块(94953 和 94954)填满了，一个块（94955）是半满的。因为有 2 个 ITL 槽位，我们需要拿 2 个满的数据块，4 个进程来模拟 ITL 死锁：

实验步骤	会话	SID	要更新的块号	要更新的行号	是否有阻塞
步骤一	1	19	94953	1	N
	2	79	94953	2	N
	3	78	94954	1	N
	4	139	94954	2	N

会话 1：

```
SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;
USERENV('SID')
--------------
            19
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2   WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
  3  AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=1;
1 row updated.
```

会话 2：

```
SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;
USERENV('SID')
--------------
            79
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2   WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
  3  AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=2;
1 row updated.
```

会话 3：

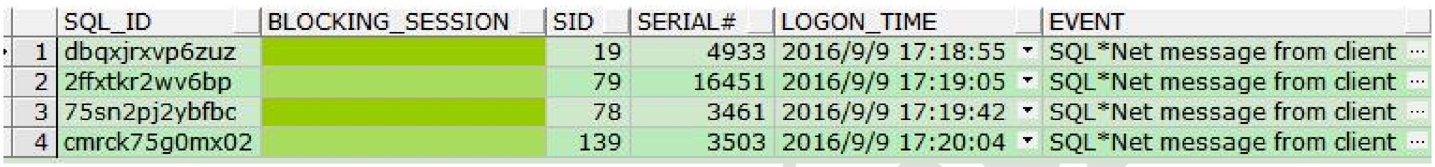
```
SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;
USERENV('SID')
--------------
            78
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2   WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
  3  AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=1;
1 row updated.
```

会话 4：

```
SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;
USERENV('SID')
--------------
           139
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2   WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
  3  AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=2;
1 row updated.
```

这个时候系统不存在阻塞，

```
SELECT NVL(A.SQL_ID, A.PREV_SQL_ID) SQL_ID,
       A.BLOCKING_SESSION,
       A.SID,
       A.SERIAL#,
       A.LOGON_TIME,
       A.EVENT
  FROM GV$SESSION A
 WHERE A.SID IN (19, 79,78,139)
 ORDER BY A.LOGON_TIME;
```



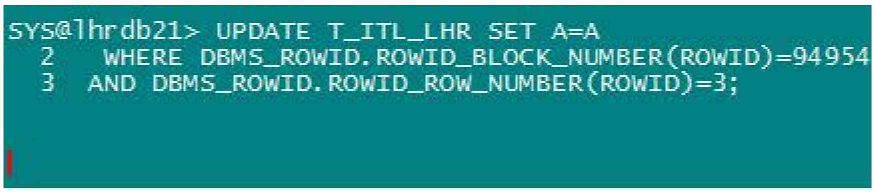
	SQL_ID	BLOCKING_SESSION	SID	SERIAL#	LOGON_TIME	EVENT
1	dbqxjrxvp6zuz		19	4933	2016/9/9 17:18:55	SQL*Net message from client
2	2ffxtkr2wv6bp		79	16451	2016/9/9 17:19:05	SQL*Net message from client
3	75sn2pj2ybfbc		78	3461	2016/9/9 17:19:42	SQL*Net message from client
4	cmrck75g0mx02		139	3503	2016/9/9 17:20:04	SQL*Net message from client

以上 4 个进程把 2 个不同块的 4 个 ITL 槽位给消耗光了，现在的情况，就是让他们互相锁住，达成死锁条件，回到会话 1，更新块 94954，注意，以上 4 个操作，包括以下的操作，更新的根本不是同一行数据，主要是为了防止出现的是行锁等待。

实验步骤	会话	SID	要更新的块号	要更新的行号	是否有阻塞
步骤一	1	19	94953 94953	1	N
	2	79		2	N
	3	78	94954 94954	1	N
	4	139		2	N
步骤二	1	19	94954	3	Y
	3	78	94953	3	Y

会话 1：

```
UPDATE T_ITL_LHR SET A=A
 WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=3;
```

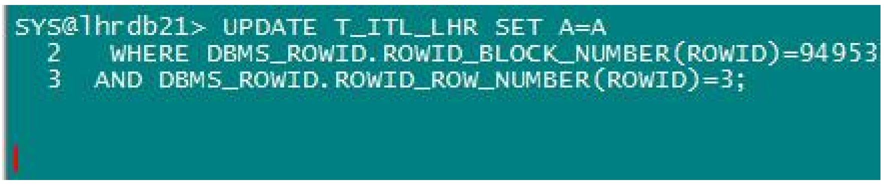


```
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2    WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
  3   AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=3;
```

会话 1 出现了等待。

会话 3：

```
UPDATE T_ITL_LHR SET A=A
 WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=3;
```

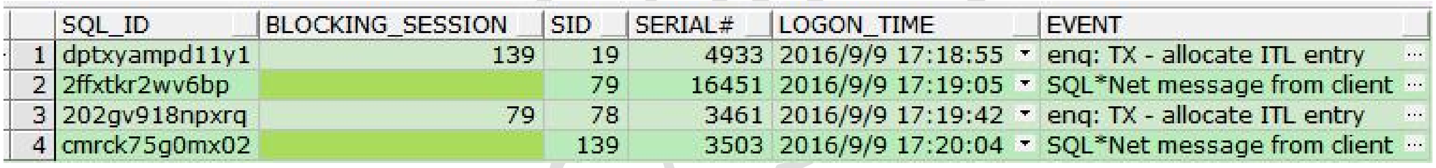


```
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2    WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
  3   AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=3;
```

会话 3 发现出现了等待。

我们查询阻塞的具体情况：

```
 SELECT NVL(A.SQL_ID, A.PREV_SQL_ID) SQL_ID,
        A.BLOCKING_SESSION,
        A.SID,
        A.SERIAL#,
        A.LOGON_TIME,
        A.EVENT
   FROM GV$SESSION A
  WHERE A.SID IN (19, 79,78,139)
  ORDER BY A.LOGON_TIME;
```



	SQL_ID	BLOCKING_SESSION	SID	SERIAL#	LOGON_TIME	EVENT
1	dptxyampd11y1	139	19	4933	2016/9/9 17:18:55	enq: TX - allocate ITL entry
2	2ffxtkr2wv6bp		79	16451	2016/9/9 17:19:05	SQL*Net message from client
3	202gv918npxrq	79	78	3461	2016/9/9 17:19:42	enq: TX - allocate ITL entry
4	cmrck75g0mx02		139	3503	2016/9/9 17:20:04	SQL*Net message from client

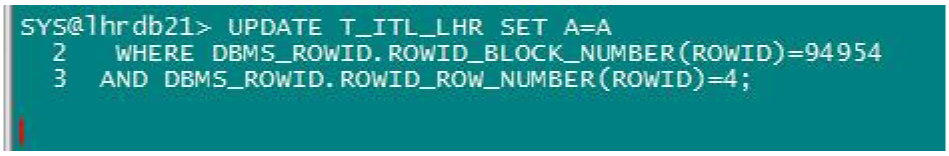
可以看到，会话 1 被会话 4 阻塞了，会话 3 被会话 2 阻塞了。

注意，如果是 9i，在这里就报死锁了，但是在 10g 里面，这个时候，死锁是不会发生的，因为这里的会话 1 还可以等待会话 4 释放资源，会话 3 还可以等待会话 2 释放资源，只要会话 2 与会话 4 释放了资源，整个环境又活了，那么我们需要把这两个进程也塞住。

实验步骤	会话	SID	要更新的块号	要更新的行号	是否有阻塞
步骤一	1	19	94953	1	N
	2	79	94953	2	N
	3	78	94954	1	N
	4	139	94954	2	N
步骤二	1	19	94954	3	Y
	3	78	94953	3	Y
步骤三	2	79	94954	4	Y
	4	139	94953	4	Y

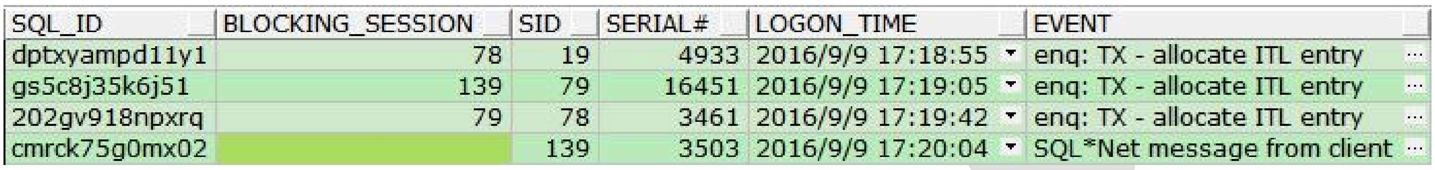
会话 2，注意，我们也不是更新的同一行数据：

```
UPDATE T_ITL_LHR SET A=A
 WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=4;
```



```
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2    WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
  3   AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=4;
```

会话 2 出现了等待，具体阻塞情况：



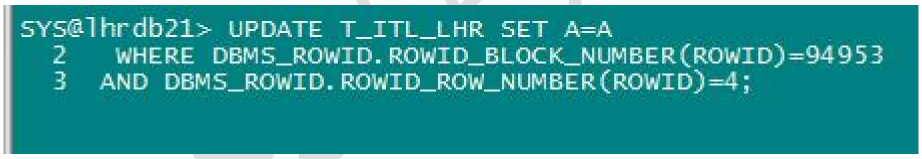
SQL_ID	BLOCKING_SESSION	SID	SERIAL#	LOGON_TIME	EVENT
dptxyampd11y1	78	19	4933	2016/9/9 17:18:55	enq: TX - allocate ITL entry
gs5c8j35k6j51	139	79	16451	2016/9/9 17:19:05	enq: TX - allocate ITL entry
202gv918npxrq	79	78	3461	2016/9/9 17:19:42	enq: TX - allocate ITL entry
cmrck75g0mx02		139	3503	2016/9/9 17:20:04	SQL*Net message from client

我做了几次实验，会话 2 执行完 SQL 后，会话 3 到这里就报出了死锁，但有的时候并没有产生死锁，应该跟系统的阻塞顺序有关，若没有产生死锁，我们可以继续会话 4 的操作。

```
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2   WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
  3  AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=3;
UPDATE T_ITL_LHR SET A=A
       *
ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource
```

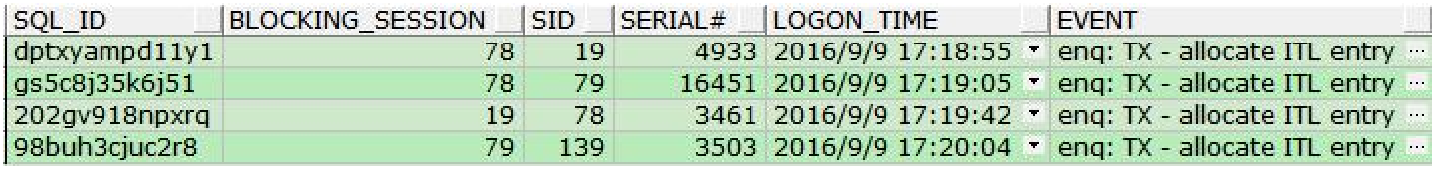
会话 4，注意，我们也不是更新的同一行数据：

```
UPDATE T_ITL_LHR SET A=A
 WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=4;
```



```
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2    WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94953
  3   AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=4;
```

会话 4 发现出现了等待。

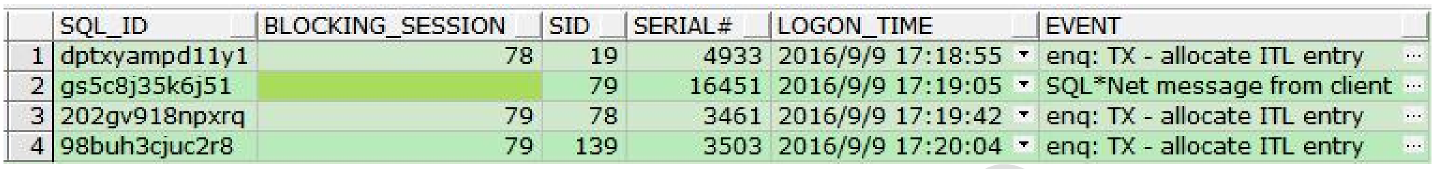


SQL_ID	BLOCKING_SESSION	SID	SERIAL#	LOGON_TIME	EVENT
dptxyampd11y1	78	19	4933	2016/9/9 17:18:55	enq: TX - allocate ITL entry
gs5c8j35k6j51	78	79	16451	2016/9/9 17:19:05	enq: TX - allocate ITL entry
202gv918npxrq	19	78	3461	2016/9/9 17:19:42	enq: TX - allocate ITL entry
98buh3cjuc2r8	79	139	3503	2016/9/9 17:20:04	enq: TX - allocate ITL entry

虽然，以上的每个更新语句，更新的都不是同一个数据行，但是，的确，所有的进程都被阻塞住了，那么，死锁的条件也达到了，等待一会（**这个时间有个隐含参数来控制的：_lm_dd_interval**），我们可以看到，会话 2 出现提示，死锁：

```
SYS@lhrdb21> UPDATE T_ITL_LHR SET A=A
  2   WHERE DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)=94954
  3  AND DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)=4;
UPDATE T_ITL_LHR SET A=A
       *
ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource
```

报出死锁之后的阻塞情况：



	SQL_ID	BLOCKING_SESSION	SID	SERIAL#	LOGON_TIME	EVENT
1	dptxyampd11y1	78	19	4933	2016/9/9 17:18:55	enq: TX - allocate ITL entry
2	gs5c8j35k6j51		79	16451	2016/9/9 17:19:05	SQL*Net message from client
3	202gv918npxrq	79	78	3461	2016/9/9 17:19:42	enq: TX - allocate ITL entry
4	98buh3cjuc2r8	79	139	3503	2016/9/9 17:20:04	enq: TX - allocate ITL entry

我们可以在会话 2 上继续执行步骤三中的 SQL，依然会产生死锁。生成死锁后，在告警日志中有下边的语句：

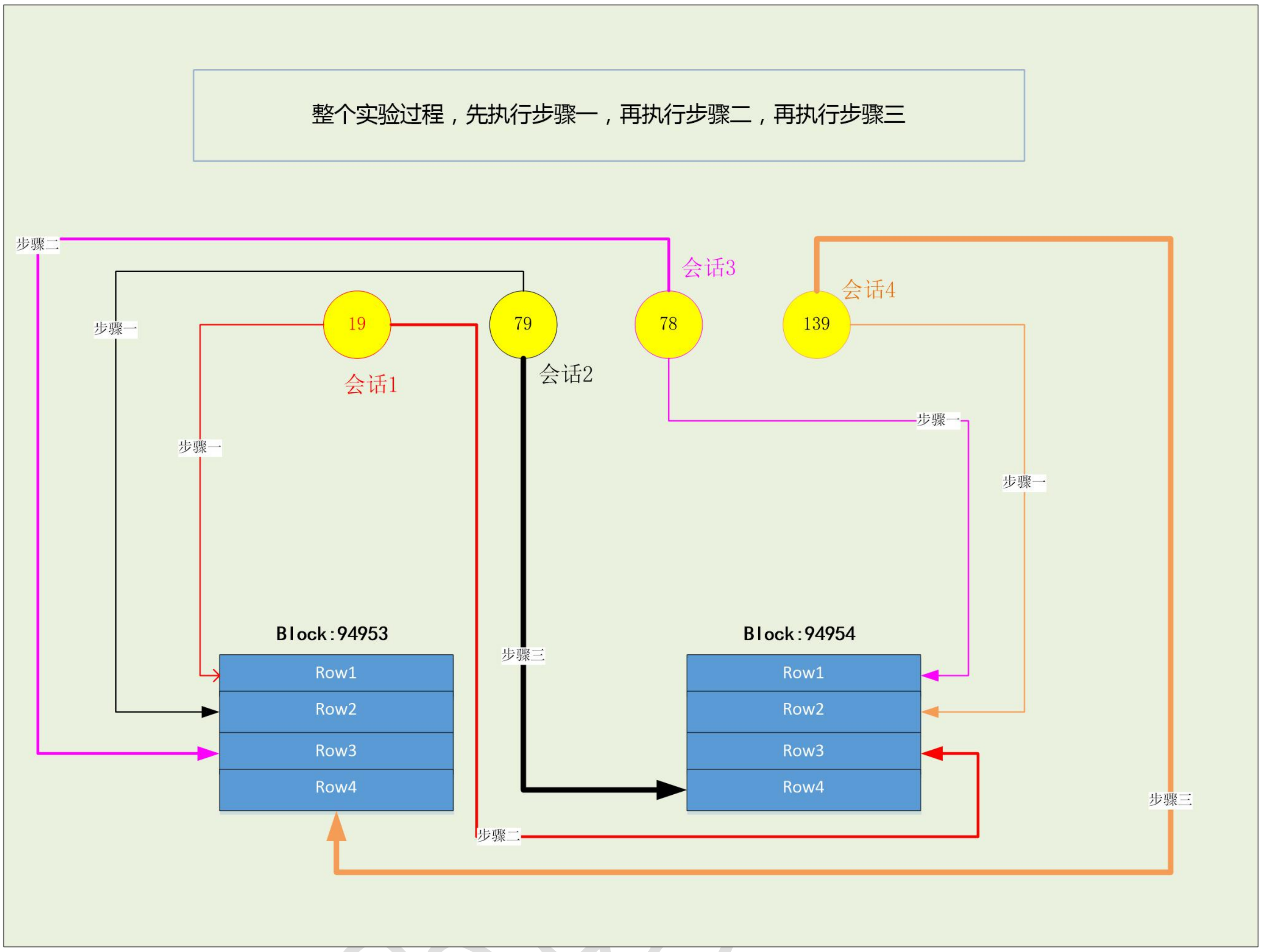
```
Fri Sep 09 17:56:55 2016
Global Enqueue Services Deadlock detected. More info in file
 /oracle/app/oracle/diag/rdbms/lhrdb2/lhrdb21/trace/lhrdb21_lmd0_17039368.trc.
```

其中的内容有非常经典的一段 Global Wait-For-Graph(WFG)：

```
*** 2016-09-09 17:48:22.216
Submitting asynchronized dump request [1c]. summary=[ges process stack dump (kjdglblkrdm1)].
Global blockers dump end:-----------------------------------
Global Wait-For-Graph(WFG) at ddTS[0.395] :
BLOCKED 0x700010063d59b90 3 wq 2 cvtops x1001 TX 0x7000b.0xa67(ext 0x2,0x0)[1002-0029-00008387] inst 1
BLOCKER 0x700010063c6d268 3 wq 1 cvtops x28 TX 0x7000b.0xa67(ext 0x2,0x0)[1002-002D-00003742] inst 1
BLOCKED 0x700010063d5adc8 3 wq 2 cvtops x1 TX 0x30021.0x848(ext 0x2,0x0)[1002-002D-00003742] inst 1
BLOCKER 0x700010063d5a4b8 3 wq 1 cvtops x28 TX 0x30021.0x848(ext 0x2,0x0)[1002-0029-00008387] inst 1
```

至于每个参数到底是什么意思，目前还没有去研究，等待大神可以无偿解释一下。至于如何解决 ITL 产生的死锁，无非就是增大表和索引的 initrans 和 PCT_FREE 的值，可以参考本 BLOG 中的 ITL 死锁问题解决。

该实验过程可能有点复杂，小麦苗画了个图来说明整个实验过程：



1.5 与文章有关的相关连接

【推荐】 update 修改为 merge （max+decode）	http://blog.itpub.net/26736162/viewspace-1244055/
【推荐】 采用 merge 语句的非关联形式再次显神能	http://blog.itpub.net/26736162/viewspace-1222423/
【推荐】 采用 MERGE 语句的非关联形式提升性能 ---后传	http://blog.itpub.net/26736162/viewspace-1222417/
【推荐】 采用 MERGE 语句的非关联形式提升性能	http://blog.itpub.net/26736162/viewspace-1218671/

自相矛盾：一个进程可以自成死锁么	http://blog.itpub.net/26736162/viewspace-2080712/
oracle 死锁类型和原因分析	http://blog.itpub.net/26736162/viewspace-1744719/
【DEADLOCK】Oracle“死锁”模拟	http://blog.itpub.net/26736162/viewspace-1744705/
[转]:深入研究 ITL 阻塞与 ITL 死锁	http://blog.itpub.net/26736162/viewspace-2124539/

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
- QQ 群：230161599　微信群：私聊
- 本文 itpub 地址：http://blog.itpub.net/26736162/viewspace-2124771/　本文博客园地址：http://www.cnbl
- 本文 pdf 版：http://yunpan.cn/cdEQedhCs2kFz （提取码：ed9b）
- 小麦苗云盘地址：http://blog.itpub.net/26736162/viewspace-1624453/
- 联系我请加 QQ 好友(642808185)，注明添加缘由
- 于 2016-09-05 09:00~ 2016-09-10 19:00 在中行完成

手机长按下图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,免费学习最