深入解析 ORACLE 等待事件 DIRECT PATH READ

BY SHOUG.魏兴华

SH'OUG
SHANGHAI ORACLE USERS GROUP
上海ORACLE用户组

How to Find SHOUG?

传统读取数据的方式是服务器进程通过读取磁盘，然后把数据加载到共享内存中，这样后面的进程就可以通过共享内存访问这些数据，不用再通过缓慢的磁盘读取来完成。direct path read 读取数据块方式，是指服务器进程直接读取数据文件，不经过 buffer cache，这种方式读取的数据块会加载到服务器进程的 PGA 内中当中，不会进入 buffer cache 中。11G 之前的 direct path read 主要用于并行查询中，此等待事件的三个参数 p1,p2,p3 分别代表：file#：文件号，first block#：读取的起始块号，block count：以 first block 为起点，连续读取的物理块数。而从 11G 之后，direct path read 不仅可用于并行查询，在符合某些条件后，串行的全表扫描也可以利用 direct path read 方式来完成。这里主要介绍 11G direct path read 相关的特性。

有一次经历让我印象深刻，那是要对一张大表上一个核心索引进行重建，表的大小几十个 G 是有的。当时预估了时间（全表扫描+排序）应该十几分钟可以搞定，但是让我吃惊的是整个重建过程足足搞了快 1 个小时。不停的查看重新索引会话的等待事件，大多数时候都是 db file sequential read，零星的能看到几次 db file scattered read，虽然指定的多块读大小为 128，但是观察到的 p3 参数小的可怜。导致索引重建慢的原因不难分析，是由于这个表的不少数据被缓存进了 buffer cache 中，虽然设置的多块读参数为 128，但是由于总是无法读取到符合要求的连续的数据块而让性能大打折扣，这里所谓的符合要求指的是读取的数据块都不在 buffer cache 之中。例如：

上面的 12 个块为一个 extent，假设都不在 buffer cache 中的话，一次物理 IO 就可以完成，但是由于部分块被 cache，而导致总共需要 6 次的物理 IO 才行。因此在做全表扫描的情况下，表里的数据一部分被 cache 并不见得总是好事。当时我在想，如果可以绕过 buffer cache，直接路径读取就好了。11G 以后，ORACLE 终于推出了这个功能，在全表扫描的时候，如果 ORACLE 认为这个表足够大，就会启用直接路径读取，而不需要像 11G 之前，必须先把数据加载到 buffer cache。

采用 direct path read 完成读取的条件

1）表大于_small_table_threshold 的参数值设置

Oracle 通过隐含参数_small_table_threshold 来界定大表小表的临界，Oracle 认为对于大表执行直接路径读取的意义比较大，对于小表通过将其缓存可能受益更大。_small_table_threshold 的单位为 block。默认为 db cache size 的 2%大小，在实例启动过程中动态决定。11GR2 之前，表的大小要是_small_table_threshold 参数值的 5 倍才会采取直接路径读取方式，11GR2 后只需要满足_small_table_threshold 定义的大小就会采取直接路径读取。我们可以简单的看一下 11GR2 下的一个测试：

```
create tablespace test datafile '+DG_DATA/test/test.dbf' size 64m segment space management
manual;
create table t (v varchar2(100)) pctused 1 pctfree 99 tablespace test;
show parameter small


NAME                                 TYPE                   VALUE
------------------------------------ ---------------------- ------------------------------
_small_table_threshold               integer                3000


CREATE OR REPLACE FUNCTION GET_ADR_TRSH(P_STEP    IN NUMBER, P_START IN NUMBER DEFAULT 0,
```

```
P_STOP    IN NUMBER DEFAULT NULL)
    RETURN NUMBER IS
    L_PRD      NUMBER;
    L_CNT      NUMBER;
    L_BLOCKS NUMBER := 0;
    L_START    NUMBER := P_START;
BEGIN
    EXECUTE IMMEDIATE 'truncate table t';
    LOOP
      INSERT /*+ append */
      INTO T
          SELECT RPAD('*', 100, '*')
              FROM DUAL
          CONNECT BY LEVEL <= P_STEP + L_START;
      COMMIT;
      L_BLOCKS := L_BLOCKS + P_STEP + L_START;
      L_START    := 0;
      EXECUTE IMMEDIATE 'alter system flush buffer_cache';
      SELECT /*+ full(t) */
         COUNT(*)
          INTO L_CNT
          FROM T;
      SELECT VALUE
          INTO L_PRD
          FROM V$SEGMENT_STATISTICS
         WHERE OWNER = USER
           AND OBJECT_NAME = 'T'
           AND STATISTIC_NAME = 'physical reads direct';
      EXIT WHEN(L_PRD > 0 OR L_BLOCKS > NVL(P_STOP, L_BLOCKS));
    END LOOP;
    RETURN L_BLOCKS - P_STEP;
END;
/


set serveroutput on
DECLARE
    L_TRSH NUMBER;
BEGIN
    L_TRSH := GET_ADR_TRSH(10, 1500, 4000);
    DBMS_OUTPUT.PUT_LINE(L_TRSH);
END;
/
3000
```

以上的代码大家可以研究一下，我不做详细说明了，大体的意思是不断的增加表大小，观察增加到表上的 block 有多少时，会采取直接路径读取。我们测试的结果是 3000，刚好和我们定义的_small_table_threshold 参数值完全吻合。11GR1 的话，经过测试是_small_table_threshold 的 5 倍的时候才会出现直接路径读取。但是表的大小并不仅仅是唯一决定是否采用直接路径读取的因素，还有其他两个因素：脏块的比例、表中数据被 cache 的比例

2）表上的脏块小于表总 block 数的 25%

下面的代码通过不断增加表的脏块数，当达到表脏块数的 25%的时候，不在发生直接路径读取事件了。

```
CREATE OR REPLACE FUNCTION GET_DIRTY_TRSH(P_STEP    IN NUMBER, P_START IN NUMBER DEFAULT 0,
P_STOP    IN NUMBER DEFAULT NULL)
    RETURN NUMBER IS
    L_TRSH    NUMBER := 0;
    L_PRD     NUMBER := 0;
    L_CNT     NUMBER := 0;
    L_START NUMBER := P_START;
BEGIN
    EXECUTE IMMEDIATE 'alter system flush buffer_cache';
    LOOP
      L_TRSH := L_TRSH + P_STEP + L_START;
      UPDATE T SET V = V WHERE ROWNUM <= L_TRSH;
      COMMIT;
      L_START := 0;
      SELECT /*+ full(t) */
         COUNT(*)
          INTO L_CNT
          FROM T;
      SELECT VALUE
          INTO L_CNT
          FROM V$SEGMENT_STATISTICS
         WHERE OWNER = USER
           AND OBJECT_NAME = 'T'
           AND STATISTIC_NAME = 'physical reads direct';
      EXIT WHEN L_CNT = L_PRD OR L_TRSH > NVL(P_STOP, L_TRSH);
      L_PRD := L_CNT;
    END LOOP;
    RETURN L_TRSH;
END;
/


DECLARE
    L_TRSH NUMBER;
BEGIN
    L_TRSH := GET_DIRTY_TRSH(1, 350, 4000);
    DBMS_OUTPUT.PUT_LINE(L_TRSH);
END;
/
739
```

我们看到脏块的比例大约达到表块的 25%（共 3000 个表块）的时候，直接路径读消失了。

3）表中的块被 cache 的比例小于 50%的时候

```
CREATE OR REPLACE FUNCTION GET_CACHED_TRSH(P_START IN NUMBER DEFAULT 0, P_STEP    IN NUMBER
DEFAULT 1)
    RETURN NUMBER IS
    CURSOR L_CUR IS
      SELECT /*+ index(t i_t) */
         t
          FROM T WHERE
v='*********************************************************************************************
*******';
    L_V          VARCHAR2(100);
```

```
    L_TRSH     NUMBER := 0;
    L_PRD      NUMBER := 0;
    L_CNT      NUMBER := 0;
    L_START NUMBER := P_START;
BEGIN
    EXECUTE IMMEDIATE 'alter system flush buffer_cache';
    OPEN L_CUR;
    LOOP
      FOR I IN 1 .. P_STEP + L_START LOOP
          FETCH L_CUR
              INTO L_V;
      END LOOP;
      L_TRSH    := L_TRSH + P_STEP + L_START;
      L_START := 0;
      SELECT /*+ full(t) */
         COUNT(*)
          INTO L_CNT
          FROM T;
      SELECT VALUE
          INTO L_CNT
          FROM V$SEGMENT_STATISTICS
         WHERE OWNER = USER
           AND OBJECT_NAME = 'T'
           AND STATISTIC_NAME = 'physical reads direct';
      EXIT WHEN L_CNT = L_PRD OR L_CUR%NOTFOUND;
      L_PRD := L_CNT;
    END LOOP;
    CLOSE L_CUR;
    RETURN L_TRSH;
END;
/
create index i_t on t (v);
DECLARE
    L_TRSH NUMBER;
BEGIN
    L_TRSH := GET_CACHED_TRSH(500, 1);
    DBMS_OUTPUT.PUT_LINE(L_TRSH);
END;
/

1497
```

在接近 50%（共 3000 个表块）的表块被 cache 的时候，直接路径读消失了。

	11GR1	11GR2	备注
块阀值	_small_table_threshold*5	_small_table_threshold	统计信息里记录的表的 block 数目（11GR2）。 超过此阀值后。
Block cache 阀值	表块的 50%	表块的 50%	少于此阀值

脏块阀值	表块的 25%	表块的 25%	少于此阀值

满足以上条件时，Oracle 会进行直接路径读取。

Oracle 为直接路径读取设置的三个“门槛”，非常的合理：

第一个阀值：表大小，太小的表从 direct path read 中的获益太小。但是特别需要引起你的警惕，如果表上存在统计信息，那么 ORACLE 会采取表的统计信息中记录的 block 与_small_table_threshold 的设定值来做比较，而不是表的真实大小（dba_segments 中记录的值）。这可能导致一些不是你预期的情况发生。如果你的统计信息与表的真实情况差异很大，那么你应该仔细考虑可能发生什么样的结果。如果你的表没有统计信息，ORACLE 会依据表的真实大小来决定是否进行 direct path read。

第二个阀值：脏块阀值，由于 direct path read 需要出发一个段的检查点，因此脏块太多，刷新脏块可能会导致 IO 繁忙

第三个阀值：表在内存里的 cache 率，如果 cache 率很高，那么还是走传统路径更快。direct path read 的出现，需要让 ORACLE 公司的开发人员设计一个单独的结构来存储每个表有多少数据是脏数据，有多少数据被 cache。不过这个结构目前还并未暴露给我们查询。在 flush buffer cache 后，这个结构被清空。（flush shared_pool 并不会被清空）

当你预期一个查询应该会走 direct path read，但是却走了传统的路径扫描的时候，应该检查是否违背了这三个前置条件中的一个或几个。不过很多时候，当一个查询“应该”走 direct path read 但是却没有的时候，你往往无能为力，你能在数据库运行过程中修改表的数据的 cache 比例吗？不能！你能修改表的脏数据的比例吗？往往也不能，通过修改表的统计信息中的表块数，可以满足第一个条件，但是如果不满足其他两个条件，依然不能有效。你可以通过 flush buffer cache 来满足后两个条件，但往往在生产环境中不被允许。我曾经在个人微博(新浪微博：魏兴华-DBA)发布过一个关于索引创建走不上 direct path read 的微博，现在终于想明白了，不满足第三个条件，表被 cache 的内容需要低于 50%。

direct path read 的优势

1. 我认为非常重要的，参照我文章开头举得例子，采用直接路径读取后，总能保证读取的块数是多块读参数设置的大小，提高了读取的效率
2. 大大的降低了对于 latch 的使用，进而避免了可能导致的 latch 竞争（cbc latch 等）
3. 降低了全表扫描对 buffer cache 的冲击
4. 降低了 buffer pin 的开销，有可能降低 buffer busy waits 等相关等待

如何控制 direct path read 的开关

有两种方法来启用、禁用这个特性。event 10949 或者设置隐含参数_serial_direct_read。两种方式都可以在 session 或 system 级别设置。

1. event 10949 设置后，可以禁用 direct path read。

每次测试前为了避免脏块和缓冲块对测试的影响，都先将 buffer cache 清空。我们在 session 级别测试，system 级别的测试方法是一样的，这里就不再赘述。

```
SQL> alter system flush buffer_cache;
System altered.

SQL> ALTER session SET EVENTS '10949 TRACE NAME CONTEXT FOREVER';

Session altered.

SQL> execute snap_events.start_snap
```

```
PL/SQL procedure successfully completed.


SQL> select count(*) from t;

  COUNT(*)
----------
  10090944

SQL> set serveroutput on
SQL> execute snap_events.end_snap
---------------------------------------------------------
SID:  2929:DLSP - oracle
Session Events - 08-Sep 10:58:50
Interval:-      18 seconds
---------------------------------------------------------
Event                        Waits   Time_outs        Csec     Avg Csec     Max Csec
-----                        -----   ---------        ----     --------     --------
Disk file operations I/O        17           0           0         .012            0
db file scattered read       1,115           0          96         .086            1
SQL*Net message to client        6           0           0         .000            0
SQL*Net message from client      6           0       1,323      220.444        2,350

SQL> alter system flush buffer_cache;


System altered.
SQL> ALTER session SET EVENTS '10949 TRACE NAME CONTEXT    off';

Session altered.

SQL> execute snap_events.start_snap


PL/SQL procedure successfully completed.


SQL> select count(*) from t;


  COUNT(*)
----------
  10090944


SQL> execute snap_events.end_snap
---------------------------------------------------------
SID:  2929:DLSP - oracle
Session Events - 08-Sep 10:59:56
Interval:-      8 seconds
---------------------------------------------------------
Event                        Waits   Time_outs        Csec     Avg Csec     Max Csec
-----                        -----   ---------        ----     --------     --------
```

```
db file sequential read          1            0           0        .024          0
direct path read              1113            0         300        .232          1
SQL*Net message to client        5            0           0        .001          0
SQL*Net message from client      5            0         445      89.034      2,673
```

我们通过 snap_events 脚本获取了会话执行语句前后等待事件的差值，启用 10949 事件后，执行了 db file scattered read，关闭 10949 事件后，以 direct path read 方式进行了表扫描。

2.通过设置隐含参数_serial_direct_read 来设置是否启用 direct path read，同样我们只在 session 级别做测试，system 级别可以由读者自己来完成。

```
SQL>alter session set "_serial_direct_read"=never;
Session altered.

SQL> execute snap_events.start_snap

PL/SQL procedure successfully completed.

SQL> select count(*) from t;

    COUNT(*)
----------
    10090944
SQL>execute snap_events.end_snap
SQL> ---------------------------------------------------------
SID:  2641:DLSP - oracle
Session Events - 08-Sep 10:49:46
Interval:-      9 seconds
---------------------------------------------------------
Event                        Waits   Time_outs        Csec    Avg Csec    Max Csec
-----                        -----   ---------        ----    --------    --------
Disk file operations I/O         6           0           0        .001           0
db file sequential read          1           0           0        .030           1
db file scattered read       1,115           0         154        .138           4
SQL*Net message to client        5           0           0        .000           0
SQL*Net message from client      5           0         228      45.510       1,912

PL/SQL procedure successfully completed.

SQL> alter session set "_serial_direct_read"=auto;

Session altered.

SQL> alter system flush buffer_cache;

System altered.

SQL> execute snap_events.start_snap

PL/SQL procedure successfully completed.

SQL> select count(*) from t;
```

```
    COUNT(*)
----------
    10090944
SQL>execute snap_events.end_snap
SQL> ---------------------------------------------------------
SID:  2641:DLSP - oracle
Session Events - 08-Sep 10:50:21
Interval:-      10 seconds
---------------------------------------------------------
Event                         Waits   Time_outs        Csec    Avg Csec     Max Csec
-----                         -----   ---------        ----    --------     --------
db file sequential read           1           0           0        .028            1
direct path read               1112           0          42        .164            0
SQL*Net message to client         5           0           0        .000            0
SQL*Net message from client       5           0         730     146.016        1,912

PL/SQL procedure successfully completed.
```

_serial_direct_read 设置为 never 后，系统采取了传统的扫描路径，出现了 db file scattered read 等待时间，_serial_direct_read 为 auto 后，采用了 direct path read 的等待事件。

direct path read 可能的副作用

1. 会发生段一级的检查点（后面详细介绍），因此在查询真正开始执行前，会做这个额外的准备工作。而且可能会造成 IO 抖动，因为要写脏数据。
2. 如果你的表需要频繁的全表扫描读取，还是用传统的读取方式比较好。
3. 在 MOS 中搜索 direct path read，会发现它可能会导致多次的延迟块清除（后面详细介绍）

直接路径读为什么要发生检查点？

Oracle 的一致性读取代表的是：读取到的数据是查询开始时刻的数据，在你对一个大表发出查询前，首先需要发生一个段级别的检查点来保证这个段上的脏数据已经被刷新到了磁盘，如果不这样做会发生什么情况？举个例子就会很容易明白：查询发生时，某条记录的 id 值在磁盘上的值为 5，内存中的值为 6。如果不发生段一级的检查点就开始直接路径读取，那么进程在读取到这个记录所在的数据块后发现，块上的 scn 是小于查询 scn 的，读取是安全的，因此直接把 5 作为结果返回给用户。完全违背了数据库（块）的一致性读取。因此 Oracle 在直接路径读取之前，都会发生一个段一级的检查点来保证一致性的读取。通过 10046 很容易看到发生的段级别的检查点。

```
PARSING IN CURSOR #4573489040 len=22 dep=0 uid=45 oct=3 lid=45 tim=1735265030575 hv=2763161912
ad='70000036058fd08' sqlid='cyzznbykb509s'
select count(*) from t
END OF STMT
PARSE #4573489040:c=60,e=101,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=1842905362,tim=1735265030574
EXEC #4573489040:c=51,e=84,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=1842905362,tim=1735265030719
WAIT #4573489040: nam='SQL*Net message to client' ela= 9 driver id=1650815232 #bytes=1 p3=0
obj#=-1 tim=1735265030759
WAIT #4573489040: nam='reliable message' ela= 89 channel context=504403173615444552 channel
handle=504403173661108944 broadcast message=504403173662264424 obj#=-1 tim=1
735265031335
WAIT #4573489040: nam='enq: KO - fast object checkpoint' ela= 10064 name|mode=1263468550
2=65588 0=1 obj#=-1 tim=1735265041430
```

说明：
1）直接路径读取如果有必要，也会去构造 CR 块，不过这里的 CR 块是在 PGA 中构造的，而不是在 buffer cache 中构造的。
2）如果表上没有任何的脏数据，并不会触发段上的检查点，这一点也很容易用 10046 跟踪出来，因此如果你想通过 10046 跟踪到段检查点，需要保证这个表上在内存中有脏数据。

延迟块清除与直接路径读

由于直接路径读取是发生在进程的 PGA 中的，如果读取过程中，ORACLE 发现一些块没有做块清除，会在 PGA 中进行延迟块清除的操作，但是这个清除的操作并不会记录日志，且被清除过的块并不会被刷回到磁盘。正是由于延迟清除过的块不被写回到磁盘，因此如果有比较多的进程来进行直接路径读取，就会导致各个进程反复的进行块清除的操作，一定程度上浪费了 CPU 资源。（我们下面的测试用到的 snap_my_stats 包，可以采样两次会话的统计信息差值，来获取两次采样间的信息增量）
1）我们先看看传统路径扫描下，延迟块清除的情况
update t set object_id=1 where rownum<10000;

另开一个 session 对 buffer_cache 进行刷新。
SQL> commmit;

禁用 direct path read
SQL> alter session set "_serial_direct_read"=never;

```
Session altered.

SQL> execute snap_my_stats.start_snap


PL/SQL procedure successfully completed.

SQL> select count(*) from t;

  COUNT(*)
----------
  10090944

SQL> execute snap_my_stats.end_snap
SQL> ---------------------------------
Session stats - 08-Sep 11:51:00
Interval:-  11 seconds
---------------------------------
Name                                      Value
----                                      -----
redo size                                 9,764
cleanouts only - consistent read gets       135
immediate (CR) block cleanout applications  135
commit txn count during cleanout            135
cleanout - number of ktugct calls           135


PL/SQL procedure successfully completed.
```

可以看到，相关延迟块清除的工作，都有了相应的指标，而且清除过程中产生了 redo。我们来看看第二次读取还会不会发生延迟块清除。

```
SQL> execute snap_my_stats.start_snap

PL/SQL procedure successfully completed.

SQL> select count(*) from t;
```

```
    COUNT(*)
----------
    10090944

SQL> execute snap_my_stats.end_snap
SQL> ---------------------------------
Session stats - 08-Sep 11:51:24
Interval:-  3 seconds
---------------------------------
Name                                                  Value
----                                                  -----
DB time                                                 343
non-idle wait count                                       5
consistent gets                                     140,109
consistent gets from cache                          140,109

PL/SQL procedure successfully complete
```

没有任何延迟块清除的统计信息了

采用传统的路径读取，延迟块清除只需要进行一次，后续如果再有会话读取，不需要再做任何有关块清除的工作。

2）直接路径扫描下延迟块清除的情况

准备工作略（update,flush buffer cache，commit）

```
SQL> alter session set "_serial_direct_read"=auto;

Session altered.

SQL> execute snap_my_stats.start_snap


PL/SQL procedure successfully completed.


SQL> select count(*) from t;

    COUNT(*)
----------
    10090944

SQL>execute snap_my_stats.end_snap
SQL> ---------------------------------
Session stats - 08-Sep 11:53:39
Interval:-  4 seconds
---------------------------------
Name                                                Value
----                                                -----
cleanouts only - consistent read gets                 135
immediate (CR) block cleanout applications            135
commit txn count during cleanout                      135
cleanout - number of ktugct calls                     135

PL/SQL procedure successfully completed.
```

第一次读取发生了延迟块清除。但是由于 redo size 没发生任何变化，因此没有结果显示。

```
SQL> execute snap_my_stats.start_snap


PL/SQL procedure successfully completed.
```

```
SQL> select count(*) from t;

    COUNT(*)
----------
    10090944

SQL>execute snap_my_stats.end_snap
SQL> ---------------------------------
Session stats - 08-Sep 11:53:52
Interval:-  3 seconds
---------------------------------
Name                                                Value
----                                                -----
cleanouts only - consistent read gets                 135
immediate (CR) block cleanout applica                 135
commit txn count during cleanout                      135
cleanout - number of ktugct calls                     135


PL/SQL procedure successfully completed.
```

第二次查询依然会进行延迟块清除，一定程度上多消耗了 CPU 时间。

参考文章：

http://afatkulin.blogspot.com/2009/01/11g-adaptive-direct-path-reads-what-is.html
http://afatkulin.blogspot.com/2012/07/serial-direct-path-reads-in-11gr2-and.html

补充：

表中的数据块被 cache 的数量，可以从 **X$KCBOQH.NUM_BUF 来查看。**

11GR2 依据统计信息来决定表的大小。通过隐含参数 _*direct_read_decision_statistics_driven* 来控制。
http://blog.tanelpoder.com/2012/09/03/optimizer-statistics-driven-direct-path-read-decision-for-full-table-scans-_direct_read_decision_statistics_driven/

如果是 online 创建索引，即使指定并行，也无法使用 direct path read，具体原因还不清楚。

ORACLE 在线操作功能：9I/10G 11G online index 的实现过程分析

ORACLE 的 ONLINE 创建索引在很多手册上被描述为：不阻塞写的安全操作。但是 11G 前，ORACLE 一直没能真正做到这一点。MYSQL，POSTGRES 里好像也一直没能实现 ORACLE ONLINE 的功能，他们的创建都会阻塞写操作。我们这里分析下 ORACLE ONLINE 创建索引的步骤。

9I、10G 版本下 ONLINE REBUILD 创建索引的过程。

创建索引前，开启如下跟踪事件，第一个跟踪事件，可以观察到锁的申请情况。第二个跟踪事件可以观察到递归执行的 SQL。

```
alter session set events '10704 trace name context forever,level 10';
alter session set events '10046 trace name context forever,level 12';
```

ORACLE ONLINE REBUILD 索引的思路就是通过：索引创建开始时候的数据+MRGEE 索引创建过程中的增量来完成整个创建。

索引创建开始时候的数据：通过一致性读来获得索引创建过程中的增量：通过 IOT（索引组织表）表记录的变化日志

来实现。
整个过程分为如下几个步骤：

1.申请表上类型为 2 的 TM 锁。创建 IOT 表，用来捕获增量。

```
1. 跟踪文件摘要：
2. *** 2010-07-27 23:07:16.000
3. ksqcmi: TM,18fe,0 mode=2 timeout=21474836
4. ksqcmi: returns 0
5. create table "SYS_JOURNAL_6399" (C0 NUMBER, opcode char(1), partno number, rid rowid, primary key( C0 , rid ))
6. organization index TABLESPACE "USERS"
7. CREATE UNIQUE INDEX "SYS_IOT_TOP_6406" on "SYS_JOURNAL_6399"("C0","RID") INDEX ONLY TOPLEVEL TABLESPACE "USERS" NOPARALLEL
```

创建 IOT 表的时候，持有的是原表上 2 的 TM 锁。这是个非常低级别的锁。即使此时，表上有未提交事务，也不会阻塞这个操作，因为 DML 操作获得的 3 类型的 TM 锁与 2 类型的 TM 锁具有兼容性。

2.将类型为 2 的锁转化为类型为 4 的 TM 锁。为了阻塞后续的 DML 操作。
ORACLE 在 ONLIEN 创建索引过程中，通过一致性读取表段+IOT 表增量 来完成整个创建。IOT 表记录了，一致性读取表数据后的发生的所有数据变化。因此在真正开始一致性拷贝旧数据之前，需要确保表上没有事务。可以设想一下，如果 ORACLE 不保证表上所有的事务已结束就开始一致性读取，那么一致性读取表数据建立的索引段+IOT 表 MERGE 完成后，就会丢失数据。IOT 表只记录了一致性读取表数据后的所有数据变化。对于之前的变更不做记录。
举个例子：数据块 1 的一条记录做了修改，把 A 值变为了 B 值，但是这个事务没提交，如果 ORACLE 不等待这个事务结束就开始创建索引，那么一致性读取数据块的内容，读取到的将是内容 A。在创建索引过程中，这个事务提交了，但是这个 B 并不会进入到 IOT 表里，因为这个事务是索引创建开始前开始的，IOT 表不记录。最终索引创建完成后，这个 B 会丢失。

因此 ORACLE 在这个步骤里申请了一个等级稍微高点的 TM 4 锁，假如表上存在事务，那么这个 4 的 TM 锁将不能获得，因为它与事务的 TM 3 锁冲突。持有这个 TM 4 锁，还为了阻塞后面的 DML 操作，为了让索引重建尽快可以开始。而这个持有的 TM 4 锁，经常是我们在 ONLINE 创建索引过程中，风险最大的地方。因为这个 4 锁一旦获得不了，它将会阻塞后续的所有的 DML 操作。

```
1. 跟踪文件摘要：
2. *** 2010-07-27 23:07:16.000
3. ksqcmi: TM,18fe,0 mode=4 timeout=21474836
4. WAIT #1: nam='enqueue' ela= 3072242 p1=1414332420 p2=6398 p3=0
```

3.因为 DML 操作都获得的是 TM 为 3 的锁，它跟 4 TM 锁不兼容，因此一旦可以获取到类型为 4 的 TM 锁，代表表上事务都已经结束，这个时候，就可以释放这个 TM 4 锁，将锁降级为类型为 2 的 TM 锁。至此，开始读取表数据建立索引。需要注意的是，这里的读取表数据，是一致性读取。
建立新索引的时间要依据表的大小。整个过程要做的事情就是一致性读取表数据+排序

```
1. 跟踪文件摘要：
2. *** 2010-07-27 23:07:41.000
3. ksqcmi: TM,18fe,0 mode=2 timeout=21474836
4. ksqcmi: returns 0
```

4.读取 IOT 表上的变更，在新的索引段上进行 MERGE.这个过程中会发现 IOT 表的记录数逐渐在减少。最终完成时，只剩下未提交事务所修改的记录保留在 IOT 表里。类似如下的情况。由于这些记录还没提交，ONLIEN 进程就只能等待。

```
1. select * from SYS_JOURNAL_207626;
2. C0 OP PARTNO RID
3. ---------- -- ---------- ------------------
4. 11 I 0 D/////AAGAAAe4DAAa
5. 11 I 0 D/////AAGAAAe4DAAb
```

```
6.  12 D 0 D/////AAGAAAe4DAAa
7.  12 D 0 D/////AAGAAAe4DAAb
8.  12 D 0 D/////AAGAAAe4DAAc
9.  12 D 0 D/////AAGAAAe4DAAd
```

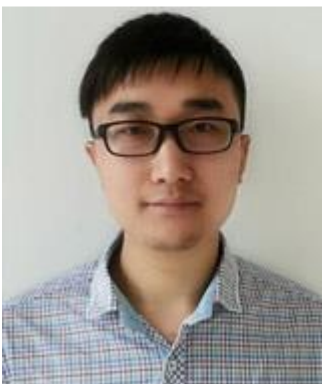
5.获取表上类型为 4 的 TM 锁，等待未提交的事务结束，这个等待阻塞后续的事务，以免让索引重建过程“无限期”的等待。这里又是一个风险点，因为申请了一个 TM 4 锁，会阻塞后面的 DML 操作。
一旦步骤 4 里涉及的变更记录都已经提交，那么就会把最终的这批记录 MERGE 进新索引里。最后删除 IOT 表。整个过程完毕。

有个疑问，就是在进行 MERGE 过程中,ORACLE 依据什么判断 IOT 表里那些记录已经提交，那些没提交？IOT 表里貌似没这个字段。看来 ORACLE 9I、10G 所谓的 ONLINE 其实并不安全，因为步骤 2、5，会进行一个锁转化，而且是低级锁向高级锁的转换，这个锁一旦获取不了，就会导致 DML 操作的阻塞。如果此时表上有大并发的事务，那么你的应用连接池可能会被撑爆。

11G 后，这个情况完全变了，ORACLE 做到了真正的 ONLINE.11G ONLINE REBUILD 创建索引的过程。
1.申请表上类型为 2 的 TM 锁，创建 IOT 表。
2.创建完 IOT 表后，如果表上存在事务，那么会发生等待，不过这个发生的等待是 TX 事务等待，不是在表上的 TM 等待了。
3.创建索引进程会被未提交的事务挂起，但是这个时候存在 DML 操作的话，可以正常进行。但是这些 DML 操作变更并不会记录进 IOT 表里。
4.如果表上事务都已经提交，那么进行一致性读取表上的旧数据的过程，然后对数据排序。例如：如果表上存在 3 个没有提交的事务，那么会先等待事务 1，事务 1 提交后，会等待事务 2，事务 2 提交后，会等待事务 3.直到事务 3 提交，那么才可以进行下一步。这个过程可以通过观察 v$session 的 P1,P2 参数来了解，随着事务的提交，P1,P2 的值也会发生变化，P1,P2 的值反应了当前等待在哪个事务上。这个过程虽然不会阻塞事务，但是我们不难发现，如果表上事务并发比较大，那么重建索引的进行就会被延期。知道表上的“那一刻”不存在事务。
5.把 IOT 表里的数据在新索引段上进行 MERGE,这个过程中，会发现 IOT 表的数据逐渐减少。最终只剩下没提交事务的数据。
6.等待 IOT 表里未提交的事务进行提交，这个时候后台会等待 TX 事务锁，类型为 4.直到所有的事务都已经提交，这个过程才会完成。这个
过程不会阻塞新进的事务，ORACLE 会把变更的数据记录进 IOT 表里（当然要涉及到跟索引字段相关的数据才会被记录进 IOT 表里）。

11G 最大的特点是不需要对 TM 锁进行锁转化了，11G 前由于这种锁转化，可能会阻塞 DML 操作，可能会引起故障。但是理论上 11G 这种做法也会引起问题，因为在第 4、6 步，在等待事务结束期间，一直有源源不断的事务进来，如果运气差的话，那么你的索引重建时间就会非常长。

作者个人简介


魏兴华

拥有 6 年 ORACLE 数据库运维经验，曾先后就职于东软集团股份有限公司、阿里巴巴（中国）网络科技有限公司，现就职于无锡新彩软件有限公司担任数 据库架构师、DBA 主管，负责公司数据库架构的设计、DB 的规模化运维和性能调优。热衷于研究 ORACLE 技术，是 OWI 性能诊断方法的提倡者，在 ITPUB 上有数篇 OWI 等待事件的精华博文，擅长从等待事件角度分析诊断 ORACLE 性能问题，对 ORACLE 的等待事件、SQL 调优、CBO 算法、 LOCK、LATCH 等技术有丰富的知识积累和研究，曾在 ORACLE 技术嘉年华、YY 分享平台、阿里巴巴等多种公开场合做过技术专题分享，并获得好评。 对执行计划的快速修正有丰富的实战经验，曾开发过迅速固定、修正执行计划的工具，此工具可以在不修改应用 SQL 的情况下，迅速固定、修正执行计划，熟练掌 握此工具，可以极大缩短 SQL 的故障时间，此脚本工具后来被阿里 DBA 团队推广使用。