回顾

在understanding - buffer cache中涉及哪些知识？

1 buffer cache 内存结构（hash bucket、chain、buffer header，buffer）

2 逻辑读

3 逻辑读与cbc latch（热块，热链）、buffer busy waits（写阻塞读）

4 物理读

5 物理读与free buffer waits（lru链表没有可用buffer）

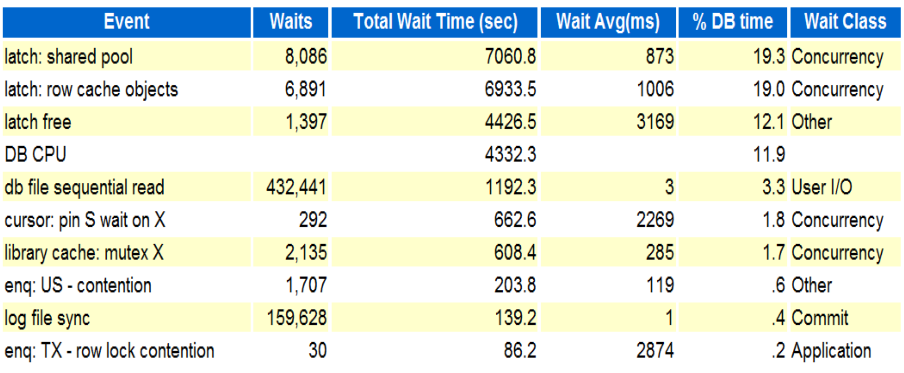
6 检查点与实例恢复

**SGA中最重要的内存区块buffer cache和shared pool**

下面了解一下shared pool涉及的知识

Oracle 引入shared pool就是为了帮助我们实现代码的共享和重用，对比pg。。

共享池，共享的东西更复杂！可能会遇到很多共享池相关问题，以下是shared pool相关的性能问题



AWR并没有专门的书介绍，我们了解了其中的概念及其相关原理，自然就读懂了

Oracle等待事件一般是ms级，latch是用于保护内存结构的低级锁，其本身也是内存，从shared pool中分配。获取latch一般是微秒级

latch：shared pool  --在分配共享池空间的时候（通常是做硬解析）发生此错误，硬解系比较多

latch：row cache objects  --row cache就是数据字典缓存，频繁修改数据字典（ddl）会发生这个错误。比较典型的就是大量使用sequence   sequence缓存20个 有100个用户在争用则会造成这个问题，我们需要加大缓存数到100等

latch free   --操作内存结构的锁等待（如cbc latch），latch是放在shared pool中的，当我们需要获取latch的时候，发现没有了，可能就是shared pool内存过小。

db cpu  一般来说我们用cpu会记录下来，包括cpu的使用时间和非空闲cpu等待时间相加，正常情况下应该排第一位

db file sequential read  物理读

cursor：pin S wait on X   sql等待做解析，同一时刻执行同一sql，发现有人正在做硬解析，那么就等待不再做硬解析，第一个人做完硬解析后，直接做软解析

library cache:mutex X  11g sql解析相关  软解析比较多

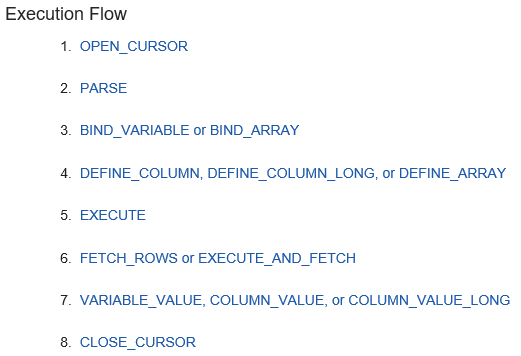
enq:TX-row lock contention   应用事物排他锁

以上是concurrent类型的都和共享池有关！

**》解读一条** **SELECT语句的执行过程**

**select salary from employees where employee\_id=:1 and department\_id=:2;**

类似 **Oracle® Database PL/SQL Packages and Types Reference>dbms\_sql**



OPEN\_CURSOR:

    指向private sql area，一块内存

PARSE

    语法：select ..from..where...  语法正确

    语义：对象的数据字典是否存在

    权限：用户权限

    转化字符ASCII  --（大小写不一致是不同的sql解析，ASCII不一致）

    执行计划：依据数据字典的统计信息，生成最优执行计划，缓存在shared pool中

BIND\_VARIABLE

   传输绑定变量（persistent area）使用同一个执行计划

DEFINE COLUMN

    把select的列定义放到cursor中

EXECUTE

    执行SQL语句（runtime area）

FETCH

    获取行数据

VARIABLE\_VALUE

    将列值读入cursor中定义的列，从buffer 拷贝到pga

CLOSE\_CURSOR

declare

v\_cursor number; --游标

v\_sql varchar2(500); --SQL语句

db\_id number := 1; --字段-id编号

db\_name varchar2(100); ---字段-姓名

v\_exe number; --在这里无意义，只是存放函数execute()返回值

v\_fet number;--在这里无意义，只是存放函数fetch\_rows()返回值

begin

v\_cursor :=dbms\_sql.open\_cursor; --打开游标

v\_sql :='select salary from hr.employees where employee\_id=:x'; --定义SQL

dbms\_sql.parse(v\_cursor,v\_sql,dbms\_sql.native); --解析动态SQL语句

dbms\_sql.bind\_variable(v\_cursor,':x',db\_id);--绑定变量把db\_id的值传给:x

dbms\_sql.define\_column(v\_cursor,1,db\_name,100);--定义列，db\_name对应SELECT 语句中的第1列

v\_exe :=dbms\_sql.execute(v\_cursor);--执行动态SQL语句

v\_fet :=dbms\_sql.fetch\_rows(v\_cursor); --获取数据

dbms\_sql.column\_value(v\_cursor,1,db\_name); --第1列的值被读入db\_name中

dbms\_output.put\_line('result:'||db\_name); --打印结果

dbms\_sql.close\_cursor(v\_cursor); --关闭游标

end;

/

result:xsj ---返回的结果

》解析

硬解析： sql第一次被执行，那么sql execution flow会执行一遍，分配内存，缓存sql文本及执行计划

软解析： sql文本及执行计划已被缓存在shared pool中，直接在内存（library cache）中找到运行

软软解析：sql文本及执行计划已被缓存在shared pool中，直接在内存（pga）中找到运行

解析访问的内存越多，相应的内存锁就会越多

查看session解析

select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic# and name like '%parse%';

NAME                                              VALUE

---------------------------------------------------------------- ----------

parse time cpu                                          9    --解析花费的cpu时间

parse time elapsed                                     8    --解析花费的总时间

parse count (total)                                   17    --总的解析次数

parse count (hard)                                     8    --硬解析次数     软=总-硬

parse count (failures)                                     0

parse count (describe)                                     0

实验： hard parse和soft parse！！！！

conn jason/jason

create table test\_parse (id number);

SQL> select a.name,b.value from v$statname a,v$mystat b where a.statistic#=b.statistic# and name like '%parse%';

NAME                                              VALUE

---------------------------------------------------------------- ----------

parse time cpu                                        20

parse time elapsed                                   27

parse count (total)                                   224

parse count (hard)                                   74

parse count (failures)                                     0

parse count (describe)                                     0

begin

  for i in 1..10

   loop

     execute immediate 'insert into test\_parse values ('||i||')';

   end loop;

   commit;

end;

/

注： ||i||  代表的是动态数值

NAME                                              VALUE

---------------------------------------------------------------- ----------

parse time cpu                                        21

parse time elapsed                                   29

parse count (total)                                   245   ==224   增加了21次

parse count (hard)                                   86    ==74   增加了12次

parse count (failures)                                     0

parse count (describe)                                     0

再次执行一遍相同的sql，但是数值变为11-20，依然是10次不同的insert

begin

  for i in 11..20

   loop

     execute immediate 'insert into test\_parse values ('||i||')';

   end loop;

   commit;

end;

/

NAME                                              VALUE

---------------------------------------------------------------- ----------

parse time cpu                                        22

parse time elapsed                                   30

parse count (total)                                   256   ==245 又增加了11次

parse count (hard)                                   97    ==86 又增加了11次   可以看出全是硬解系

parse count (failures)                                     0

parse count (describe)                                     0

下面我们把动态数值变为绑定变量，数值变为21-30，依然是10次不同的insert

begin

  for i in 21..30

   loop

     execute immediate 'insert into test\_parse values (:1)' using i;

   end loop;

   commit;

end;

/

NAME                                              VALUE

---------------------------------------------------------------- ----------

parse time cpu                                        23

parse time elapsed                                   30

parse count (total)                                   260   ==256   又增加了4次

parse count (hard)                                   99    ==97     又增加了2次   硬解析明显减少

parse count (failures)                                     0

parse count (describe)                                     0

再执行一次

begin

  for i in 31..40

   loop

     execute immediate 'insert into test\_parse values (:1)' using i;

   end loop;

   commit;

end;

/

NAME                                              VALUE

---------------------------------------------------------------- ----------

parse time cpu                                        23

parse time elapsed                                   30

parse count (total)                                   262    ==260 又增加了2次

parse count (hard)                                   100   ==99 又增加了1次   仅有一次硬解析

parse count (failures)                                     0

parse count (describe)                                     0

所以说要用绑定变量来减少硬解析

实验： 软软解析

满足以下两个条件才可以使用软软解析

同样一条sql需要被执行3次以上才会被缓存在pga

开启参数session\_cached\_cursors

session\_cached\_cursors               integer     50

在我的session下执行sql，如果sql执行超过了3次，则把缓存的sql语句的句柄（执行计划地址指针）记录到pga中，最多可以放置50条，可以改为大数值，同样会占用多的pga空间。0表示不缓存

查看视图v$open\_cursor可以观察软软解析，信息是pga中的

select sid,sql\_text,cursor\_type from v$open\_cursor where sql\_text like '%select \*%';

那么下面我们执行一个sql

select \* from test\_parse;

SQL> select sid,sql\_text,cursor\_type from v$open\_cursor where sql\_text like '%select \*%';

       SID SQL\_TEXT

---------- ------------------------------------------------------------

CURSOR\_TYPE

----------------------------------------------------------------

     38 select \* from test\_parse

OPEN

随便执行一个sql一次，发现能够缓存在pga，但是类型为open

然后同一session再次执行另一个sql

SQL> select userenv('sid') from dual;

USERENV('SID')

--------------

         38

SQL> select sid,sql\_text,cursor\_type from v$open\_cursor where sql\_text like '%select \*%';

no rows selected

当执行另一个sql后，再次查询发现缓存的sql丢失了！！

why？？？===》原因是执行完sql，cursor没有被关闭，只是软关闭。当执行另外一个sql就把上一个sql的cursor关闭了。释放了内存。

下面我们执行同一个sql 超过3次，然后执行其他sql

select \* from test\_parse;

select \* from test\_parse;

select \* from test\_parse;

select \* from test\_parse;

select \* from TEST\_PARSE;

SQL> select sid,sql\_text,cursor\_type from v$open\_cursor where sql\_text like '%select \*%';

       SID SQL\_TEXT

---------- ------------------------------------------------------------

CURSOR\_TYPE

----------------------------------------------------------------

     38 select \* from test\_parse

SESSION CURSOR CACHED

再次查询我们发现sql依然被缓存在了pga中，且类型为session cursor cached

》Orcle 11G 新特性 result buffer cache

SQL> show parameter result

NAME                     TYPE     VALUE

------------------------------------ ----------- ------------------------------

client\_result\_cache\_lag          big integer 3000

client\_result\_cache\_size         big integer 0

result\_cache\_max\_result          integer     5

result\_cache\_max\_size             big integer 1600K

result\_cache\_mode             string     MANUAL

result\_cache\_remote\_expiration         integer     0

SQL>

result\_cache\_max\_size = 0 表示禁用

result\_cache\_max\_result =5    --单个缓存结果占结果缓存的最大比例

result\_cache\_mode

Values:

MANUAL

The ResultCache operator is added only when the query is annotated (that is, hints).

FORCE

The ResultCache operator is added to the root of all SELECT statements (provided that it is valid to do so).

注意：不建议使用force，可能会产生显著的性能和latch开销

使用NO\_RESULT\_CACHE hint会优先于参数设置。

conn system/oracle

set autotrace on;

select employee\_id,first\_name,salary from hr.employees;

Execution Plan

----------------------------------------------------------

Plan hash value: 72609621

--------------------------------------------------------------------------------

| Id  | Operation       | Name      | Rows  | Bytes | Cost (%CPU)| Time     |

--------------------------------------------------------------------------------

|   0 | SELECT STATEMENT   |           |   107 |  1605 |     3     (0)| 00:00:01 |

|\*  1 |  FILTER        |           |       |       |        |           |

|   2 |   TABLE ACCESS FULL| EMPLOYEES |   107 |  1605 |     3     (0)| 00:00:01 |

--------------------------------------------------------------------------------

Statistics

----------------------------------------------------------

    622  recursive calls

     18  db block gets

    698  consistent gets

      0  physical reads

select /\*+ result\_cache \*/ employee\_id,first\_name,salary from hr.employees;

    293  consistent gets

      0  physical reads

select /\*+ result\_cache \*/ employee\_id,first\_name,salary from hr.employees;

Execution Plan

----------------------------------------------------------

Plan hash value: 72609621

--------------------------------------------------------------------------------------------------

| Id  | Operation        | Name             | Rows  | Bytes | Cost (%CPU)| Time     |

--------------------------------------------------------------------------------------------------

|   0 | SELECT STATEMENT    |                 |   107 |  1605 |     3   (0)| 00:00:01 |

|   1 |  RESULT CACHE        | aphttrh71dbfz0jzf7aaz05s0d |     |     |          |      |

|\*  2 |   FILTER        |                 |     |     |          |      |

|   3 |    TABLE ACCESS FULL| EMPLOYEES          |   107 |  1605 |     3   (0)| 00:00:01 |

--------------------------------------------------------------------------------------------------

Statistics

----------------------------------------------------------

     10  recursive calls

     17  db block gets

      4  consistent gets

      0  physical reads

X$KSMSP 名称含义: [K]ernal [S]torage [M]emory Management [S]GA Hea[P]每一行代表 Shared Pool中的一个chunk

以下是x$ksmsp 结构:

SQL> desc x$ksmsp

SQL> desc x$ksmsp

Name                                                           Null?    Type

----------------------------------------------------------------------------------------------------------------- -------- ----------------------------------------------------------------------------

ADDR                                                                RAW(8)

INDX                                                                NUMBER

INST\_ID                                                            NUMBER

KSMCHIDX                                                            NUMBER

KSMCHDUR                                                            NUMBER

KSMCHCOM                                                            VARCHAR2(16)

KSMCHPTR                                                            RAW(8)

KSMCHSIZ                                                            NUMBER

KSMCHCLS                                                            VARCHAR2(8)

KSMCHTYP                                                            NUMBER

KSMCHPAR                                                            RAW(8)

转储共享池

ALTER SESSION SET EVENTS 'immediate trace name LIBRARY\_CACHE level 8';