第 9 章　性能诊断与 SQL 优化

对于一个数据库系统，从应用的角度来说，通常我们最期望有良好的性能，稳定的运行。所以怎样维持一个数据库高性能稳定运行就变得非常重要，对于大多数数据库维护人员来说，直接面对的问题就是在问题出现时，需要快速发现并迅速解决数据库性能等问题，提高系统持续高效运行。

本章我们将通过一些实际生产中遇到的案例进行剖析讲解，希望大家能够从中领会到诊断性能问题的思路和方法，并对具体问题，特别是 SQL 问题进行常规处理。

9.1　使用 AutoTrace 功能辅助 SQL 优化

Oracle SQL*Plus 提供一个 autotrace 的功能，可以用于跟踪 SQL 的执行计划，收集统计信息，经常被作为 SQL 的优化工具之一被广泛使用。

9.1.1 Autotrace 功能的启用

在 Oracle10g 之前，缺省的 autotrace 功能并未打开，需要通过以下步骤手工启用该功能：

1．创建基础表

这可以通过运行$ORACLE_HOME\rdbms\admin\utlxplan 脚本完成，该脚本用于创建 plan_talbe 表：

```
SQL> connect / as sysdba
已连接。
SQL> @?\rdbms\admin\utlxplan
表已创建。
```

为了使多个用户可以共享同一个 plan_table，可以为它创建一个同义词，并授权给 Public：

```
SQL> create public synonym plan_table for plan_table;
同义词已创建。
SQL> grant all on plan_table to public ;
授权成功。
```

2．创建 plustrace 角色

这需要运行$ORACLE_HOME\sqlplus\admin\plustrce.sql 脚本：

```
SQL> @?\sqlplus\admin\plustrce
SQL>
SQL> drop role plustrace;
drop role plustrace
```

```
*
ERROR 位于第 1 行:
ORA-01919: 角色'PLUSTRACE'不存在

SQL> create role plustrace;
角色已创建
SQL> grant select on v_$sesstat to plustrace;
授权成功。
SQL> grant select on v_$statname to plustrace;
授权成功。
SQL> grant select on v_$session to plustrace;
授权成功。
SQL> grant plustrace to dba with admin option;
授权成功。
SQL> set echo off
```

3．一点增强

DBA 用户首先被授予了 plustrace 角色，然后我们可以手工把 plustrace 授予 public，这样所有用户都将拥有 plustrace 角色的权限，所有数据库用户也就拥有了使用 autotrace 功能的权限。

```
SQL> grant plustrace to public ;
授权成功。
```

完成以上步骤我们就可以使用 AutoTrace 的功能了。Autotrace 有几个常用选项，简单说明如下：

- SET AUTOTRACE OFF ---------- 不生成 AUTOTRACE 报告，这是缺省模式
- SET AUTOTRACE ON EXPLAIN ------ AUTOTRACE 只显示优化器执行路径报告
- SET AUTOTRACE ON STATISTICS -- 只显示执行统计信息
- SET AUTOTRACE ON ---------- 包含执行计划和统计信息
- SET AUTOTRACE TRACEONLY ------ 同 set autotrace on，但是不显示查询输出

在 SQL*Plus 中，autotrace 的基本输出大致类似：

```
SQL> set autotrace on
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production
Execution Plan
----------------------------------------------------------
   0       SELECT STATEMENT Optimizer=CHOOSE
   1    0    COUNT (STOPKEY)
   2    1      FIXED TABLE (FULL) OF 'X$VERSION'
```

```
Statistics
----------------------------------------------------------
          18  recursive calls
           0  db block gets
           2  consistent gets
           0  physical reads
           0  redo size
         433  bytes sent via SQL*Net to client
         503  bytes received via SQL*Net from client
           2  SQL*Net roundtrips to/from client
           0  sorts (memory)
           0  sorts (disk)
           1  rows processed
```

9.1.2 Oracle10g Autotrace 功能的增强

在 Oracle10g Release 2 中，Autotrace 的功能已经被极大加强和改变。让我们先来看一下什么地方发生了改变:

```
SQL> set autotrace on
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - Prod
Execution Plan
----------------------------------------------------------
Plan hash value: 1517457201
-------------------------------------------------------------------------------
| Id  | Operation              | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |               |     1 |    47 |     0   (0)| 00:00:01 |
|*  1 |  COUNT STOPKEY         |               |       |       |            |          |
|*  2 |   FIXED TABLE FULL| X$VERSION |     1 |    47 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<2)
   2 - filter("INST_ID"=USERENV('INSTANCE'))
Statistics
----------------------------------------------------------
```

```
          0  recursive calls
          0  db block gets
          0  consistent gets
          0  physical reads
          0  redo size
        471  bytes sent via SQL*Net to client
        400  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

注意，此时 autotrace 的输出被良好格式化，并给出关于执行计划部分的简要注释。

其实这里并没有带来新的技术，从 Oracle9i 开始，Oracle 提供了一个新的工具 dbms_xplan 用以格式化和查看 SQL 的执行计划。其原理是通过对 plan_table 的查询和格式化提供更友好的用户输出。

dbms_xplan 的调用的语法类似:

```
select * from table(dbms_xplan.display(format=>'BASIC'))
```

具体用法可以参考 Oracle 官方文档。

实际上从 Oracle9i 开始我们就经常使用如下方式调用 dbms_xplan:

```
SQL> explain plan for select count(*) from dual;
Explained.
SQL> @?/rdbms/admin/utlxplp;
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
| Id  | Operation                 |  Name       | Rows  | Bytes | Cost  |
------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |             |       |       |       |
|   1 |  SORT AGGREGATE           |             |       |       |       |
|   2 |   TABLE ACCESS FULL       | DUAL        |       |       |       |
------------------------------------------------------------------------
Note: rule based optimization
10 rows selected.
```

utlxplp.sql 脚本中正是调用了 dbms_xplan:

```
SQL> get ?/rdbms/admin/utlxplp;
……
 40* select * from table(dbms_xplan.display());
 41
```

而在 Oracle10gR2 中，Oracle 帮我们简化了这个过程，一个 autotrace 就完成了所有的输出，这是易用性上的一个进步。在使用 Oracle 的过程中，经常能够感受到 Oracle 针对用户需求或

易用性的改进，这也许是很多人喜爱 Oracle 的一个原因吧。

如果足够细心大家可能还会注意到，在 Oracle10g 中 PLAN_TABLE 不再需要创建，Oracle 缺省增加了一个字典表 PLAN_TABLE$，然后基于 PLAN_TABLE$创建公用同义词供用户使用。

使用 Autotrace 功能的另外一个好处就是，可以很容易的发现各种视图的底层基础表，这可以作为大家学习和研究 Oracle 的一个重要手段：

```
SQL> set autotrace trace explain
SQL> select * from plan_table;
Execution Plan
----------------------------------------------------------
Plan hash value: 103984305
--------------------------------------------------------------------------------
| Id  | Operation                 | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                   |     1 | 11081 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| PLAN_TABLE$ |     1 | 11081 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------------
Note
-----
   - dynamic sampling used for this statement
```

9.1.3 Autotrace 功能的内部操作

当使用 Autotrace 功能时，在数据库内部，Oracle 实际上是启动了 2 个会话（session）连接，一个 Session 用于执行查询等操作，另外一个 Session 用于记录执行计划和输出最终结果等操作。让我们一起来进一步深入了解一下 Autotrace 功能。

在启用 Autotrace 之前，注意当前只有一个用户 SESSION 连接：

```
SQL> select sid,serial#,username from v$session where username is not null;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         8          5 SYS
```

在启用 autotrace 功能后，此时，另外一个 SESSION 被创建：

```
SQL> set autotrace on
SQL> select sid,serial#,username from v$session where username is not null;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         8          5 SYS
         9         14 SYS
Execution Plan
```

```
-------------------------------------------------------------
    0          SELECT STATEMENT Optimizer=CHOOSE
    1      0     FIXED TABLE (FULL) OF 'X$KSUSE'
Statistics
-------------------------------------------------------------
              0   recursive calls
              0   db block gets
              0   consistent gets
              0   physical reads
              0   redo size
```

而且注意，这两个 SESSION 都是由一个进程（Process）衍生创建的：

```
SQL> select a.sid,a.serial#,a.username,b.pid,b.spid from v$session a ,v$process b
  2   where a.PADDR = b.addr and a.username is not null;
       SID       SERIAL# USERNAME                  PID    SPID
---------- ---------- ------------ ---------- ------------
         8          5 SYS                        9    28653
         9         14 SYS                        9    28653
Execution Plan
-------------------------------------------------------------
    0          SELECT STATEMENT Optimizer=CHOOSE
    1      0     MERGE JOIN
    2      1       FIXED TABLE (FULL) OF 'X$KSUPR'
    3      1       SORT (JOIN)
    4      3         FIXED TABLE (FULL) OF 'X$KSUSE'
```

而此处的 V$PROCESS.SPID 正是操作系统的进程号：

```
SQL> ! ps -ef|grep 28653|grep -v grep
oracle    28653 28652   0 11:03 ?        00:00:00 oracleeygle (DESCRIPTION=(LOCAL=YES) (ADDRESS=
(PROTOCOL=beq)))
```

这就是通常所说的，一个进程在数据库中可能对应多个 SESSION。通过在全局启用 10046 事件（具体使用方法可以参考本章后面章节），可以得到 Autotrace 的内部操作。设置 10046 事件可以采用如下命令（在 spfile 中设置，需要重新启动数据库后方能生效，注意应当仅在测试环境才可在全局启用）：

alter system set event='10046 trace name context forever,level 12' scope=spfile;

通过 tkprof 格式化跟踪文件：

```
[oracle@jumper udump]$ tkprof eygle_ora_28653.trc auto.log aggregate=no
TKPROF: Release 9.2.0.4.0 - Production on Fri May 12 11:17:54 2006
Copyright (c) 1982, 2002, Oracle Corporation.    All rights reserved.


[oracle@jumper udump]$ ll
```

```
total 168
-rw-r--r--     1 oracle    dba           91729 May 12 11:17 auto.log
-rw-r-----     1 oracle    dba           69254 May 12 11:04 eygle_ora_28653.trc
```

检查跟踪文件或格式化后的日志，我们可以发现以上两个 SESSION（在日志中显示位 8.5 和 9.14）的内部操作：

```
*** SESSION ID:(8.5) 2006-05-12 11:03:42.892
…….
*** SESSION ID:(9.14) 2006-05-12 11:04:33.935
```

主要的步骤有：

1．执行计划的输出

通过以下 SQL 完成信息记录：

```
insert into plan_table (statement_id, timestamp, operation, options,
   object_node, object_owner, object_name, object_instance, object_type,
   search_columns, id, parent_id, position, other,optimizer, cost, cardinality,
     bytes, other_tag, partition_start, partition_stop, partition_id,
   distribution, cpu_cost, io_cost, temp_space, access_predicates,
   filter_predicates )
values
(:1,SYSDATE,:2,:3,:4,:5,:6,:7,:8,:9,:10,:11,:12,:13,:14,:15,:16,:17,:18,:19,
   :20,:21,:22,:23,:24,:25,:26,:27)
```

通过以下 SQL 完成执行计划查询输出：

```
SELECT ID ID_PLUS_EXP,PARENT_ID PARENT_ID_PLUS_EXP,LPAD(' ',2*(LEVEL-1))
   ||OPERATION||DECODE(OTHER_TAG,NULL,'','*')||DECODE(OPTIONS,NULL,','
   ('||OPTIONS||')')||DECODE(OBJECT_NAME,NULL,'',' OF '"'||OBJECT_NAME||'"')
   ||DECODE(OBJECT_TYPE,NULL,'',' ('||OBJECT_TYPE||')')||DECODE(ID,0,
   DECODE(OPTIMIZER,NULL,'',' Optimizer='||OPTIMIZER))||DECODE(COST,NULL,','
   (Cost='||COST||DECODE(CARDINALITY,NULL,'',' Card='||CARDINALITY)
   ||DECODE(BYTES,NULL,'',' Bytes='||BYTES)||')') PLAN_PLUS_EXP,OBJECT_NODE
   OBJECT_NODE_PLUS_EXP
FROM PLAN_TABLE START WITH ID=0 AND STATEMENT_ID=:1 CONNECT BY PRIOR
ID=PARENT_ID
   AND STATEMENT_ID=:1 ORDER BY ID,POSITION
```

2．统计信息输出

主要通过以下 SQL 获取统计信息名称、编号等信息：

```
SELECT STATISTIC# S, NAME
FROM SYS.V_$STATNAME WHERE NAME IN ('recursive calls','db block gets','consistent
   gets','physical reads','redo size','bytes sent via SQL*Net to client',
   'bytes received via SQL*Net from client','SQL*Net roundtrips to/from
   client','sorts (memory)','sorts (disk)') ORDER BY S
```

通过以下查询获得输出值：

```
SELECT PT.VALUE
FROM SYS.V_$SESSTAT PT WHERE PT.SID=:1 AND PT.STATISTIC# IN (7,40,41,42,115,236,
  237,238,242,243) ORDER BY PT.STATISTIC#
```

了解了这些内部操作，更有利于我们学习和理解 Oracle 的运行机制。本书中介绍的很多方法都可以辅助大家进行进一步的深入研究，希望大家在阅读中更多注意一下方法。

9.1.4 使用 Autotrace 功能辅助 SQL 优化

曾经遇到这样一个案例，有朋友在论坛中提出帮助请求，问以下这样一条 SQL 是否可以优化：

```
SELECT * FROM sys_user
 WHERE user_code = 'zhangyong'
    OR user_code IN (SELECT grp_code FROM sys_grp WHERE sys_grp.user_code = 'zhangyong')
```

首先可以通过 SQL*Plus 的 Autotrace 功能查看该 SQL 的执行计划：

```
Execution Plan
----------------------------------------------------------
0   SELECT STATEMENT Optimizer=RULE
1   0     FILTER
2   1       TABLE ACCESS (FULL) OF 'SYS_USER'
3   1       INDEX (UNIQUE SCAN) OF 'PK_SYS_GRP' (UNIQUE)
Statistics
----------------------------------------------------------
14     recursive calls
4      db block gets
30590 consistent gets
0      physical reads
。。。。。。。
0      sorts (memory)
0      sorts (disk)
3      rows processed
```

注意到该 SQL 的逻辑读高达 30590，优化该 SQL 在根本上需要降低逻辑读。而相关数据表的记录情况如下：

```
SQL> select count(distinct user_code) from sys_grp;
COUNT(DISTINCTUSER_CODE)
------------------------
14580
SQL> select count(distinct grp_code) from sys_grp;
COUNT(DISTINCTGRP_CODE)
```

```
------------------------
300
SQL> select count(distinct user_code) from sys_user;
COUNT(DISTINCTUSER_CODE)
------------------------
15190
```

通过执行计划可以知道，该 SQL 通过全表扫描访问记录数为 15190 的 SYS_USER 表，通过索引唯一键扫描访问 PK_SYS_GRP，两者过滤（FILTER）返回结果集，全表扫描及 FILTER 操作导致了大量的逻辑读。

可以尝试通过 OR 展开、索引访问避免全表扫描和 FILTER 操作，改写后的 SQL 如下所示：

```
SELECT * FROM sys_user WHERE user_code = 'zhangyong'
UNION ALL
SELECT * FROM sys_user WHERE user_code <> 'zhangyong'
    AND user_code IN (SELECT grp_code FROM sys_grp WHERE sys_grp.user_code = 'zhangyong')
```

通过 UNION ALL 将 SQL 展开，从而避免了 FILTER 操作，表联合部分通过 NESTED LOOPS 实现。改写后的 SQL 执行计划如下所示：

```
Statistics
-----------------------------------------------------------
          0  recursive calls
          0  db block gets
        130 consistent gets
          0  physical reads
。。。。。。。
          1  sorts (memory)
          0  sorts (disk)
          3  rows processed
Execution Plan
-----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=RULE
   1   0    UNION-ALL
   2   1      TABLE ACCESS (BY INDEX ROWID) OF 'SYS_USER'
   3   2        INDEX (UNIQUE SCAN) OF 'PK_SYS_USER' (UNIQUE)
   4   1      NESTED LOOPS
   5   4        VIEW OF 'VW_NSO_1'
   6   5          SORT (UNIQUE)
   7   6            TABLE ACCESS (BY INDEX ROWID) OF 'SYS_GRP'
   8   7              INDEX (RANGE SCAN) OF 'FK_SYS_USER_CODE' (NON-UNIQUE)
   9   4        TABLE ACCESS (BY INDEX ROWID) OF 'SYS_USER'
```

```
  10      9              INDEX (UNIQUE SCAN) OF 'PK_SYS_USER' (UNIQUE)
```

通过统计信息输出可以注意到，SQL 的逻辑读从原来的 30590 降低到 130，性能得到了极大提高。同时改写后的 SQL 引入了一个排序，排序来自于这一步：

```
6   5   SORT (UNIQUE)
7   6     TABLE ACCESS (BY INDEX ROWID) OF 'SYS_GRP'
8   7       INDEX (RANGE SCAN) OF 'FK_SYS_USER_CODE' (NON-UNIQUE)
```

在 SYS_GRP 表中，user_code 是非唯一键值，在 in 值判断里，要做 sort unique 排序，去除重复值，这里的 union all 是不需要排序的。

9.2获取 SQL 执行计划的方法

在进行 SQL 诊断和优化时，通常都需要获取 SQL 的执行计划，通过执行计划来判断 SQL 的执行是否合理，那么如何来获取 SQL 的执行计划就显得非常重要了。

上一节介绍的 AutoTrace 功能是获得 SQL 执行计划的方法之一，在这一节，继续讨论 SQL 执行计划获取方法及相关诊断应用。

9.2.1 通过 V$SQL_PLAN 获得执行计划

从 Oracle9i 开始，Oracle 开始通过 V$SQL_PLAN 等视图进行 SQL 执行计划的记录，通过这个视图，可以获取正在执行中或者仍然缓存着的 SQL 执行计划，从而可以帮助我们进行实时准确的数据库诊断。在 Oracle9i 中，可以通过自定义编写的一些脚本来获取 SQL 的执行计划，如通过 HASH_VALUE（可以通过 V$SESSION 或者 V$SQL、V$SQL_PLAN 视图获得 SQL 的 HASH_VALUE）输入来获取 SQL 及其执行计划：

```
oracle@/opt/oracle/tools$./getplan_by_hashvalue.sh 3870760741


SQL_TEXT
--------------------------------------------------------------------------------
select count(uspl.numusplguid) from hy_usersubplan_log uspl,hy_serviceplan sp,hy_serviceinfo s,hy_spinfo
pvd,hy_platform pf where uspl.numsplanguid + 0 = sp.numplanguid and sp.numsvrguid = s.numsvrguid and
s.numspguid + 0 = pvd.numspguid and pvd.numptguid+ 0 = pf.numptguid and pf.vc2platformid = :1 and
pvd.vc2spcode = :2 and s.vc2service_id = :3 and s.vc2ispack = :4 and uspl.vc2enabledflag = 'Y' and
uspl.datstart <= sysdate and nvl(uspl.datend, sysdate + 1) >= sysdate and uspl.vc2userid = :5
HASH_VALUE EXECUTIONS         PER_GETS MODULE
---------- ---------- ------------ ----------------------------------
3870760741          30458               78.4 JDBC Thin Client
--------------------------------------------------------------------------------
| Operation                                | PHV/Object Name         |    Rows | Bytes|     Cost |
--------------------------------------------------------------------------------
```

```
|SELECT STATEMENT                    |----- 3870760741 ----|          |         |       28 |
|SORT AGGREGATE                      |                     |          |       1 |   269 |          |
| NESTED LOOPS                       |                     |          |       1 |   269 |       28 |
|   NESTED LOOPS                     |                     |          |       1 |   204 |        9 |
|    NESTED LOOPS                    |                     |          |       1 |   178 |        7 |
|     NESTED LOOPS                   |                     |          |       1 |   100 |        6 |
|      TABLE ACCESS BY INDEX ROWID|HY_PLATFORM             |          |       1 |    20 |        2 |
|       INDEX UNIQUE SCAN            |HYUIDX_PLATFORMID    |          |      97 |       |        1 |
|      TABLE ACCESS FULL             |HY_SERVICEINFO       |          |       1 |    80 |        4 |
|     TABLE ACCESS BY INDEX ROWID |HY_SPINFO               |          |       1 |    78 |        1 |
|      INDEX UNIQUE SCAN             |HYPK_SPINFO          |          |      97 |       |          |
|    TABLE ACCESS BY INDEX ROWID   |HY_SERVICEPLAN        |          |       2 |    52 |        2 |
|     INDEX RANGE SCAN               |HYUIDX_SERVICEPLAN   |          |       2 |       |        1 |
|   PARTITION LIST ALL               |                     |          |         |       |          |
|    TABLE ACCESS BY LOCAL INDEX R|HY_USERSUBPLAN_LOG     |          |       1 |    65 |       19 |
|     INDEX RANGE SCAN               |HYIDX_USPL_USERID    |          |      72 |       |       18 |
--------------------------------------------------------------------------------
```

该脚本的编码内容如下：

```
oracle@/opt/oracle/tools$cat getplan_by_hashvalue.sh
#!/bin/ksh

$ORACLE_HOME/bin/sqlplus -s "/ as sysdba"<<EOF
set lines 121
set pages 999
col sql_text format a80
col module format a36
col per_gets format 999999999.9
select sql_text from v\$sqltext_with_newlines where hash_value=$1 order by piece;

select hash_value,executions,buffer_gets/decode(executions,0,1,executions) as per_gets,a.module
   from v\$sqlarea a where a.hash_value=$1;

set heading off
select '--------------------------------------------------------------------------------' from dual
union all
select  '| Operation                                | PHV/Object  Name         |  Rows  | Bytes|     Cost  |'  as
"Optimizer Plan:" from dual
union all
select '--------------------------------------------------------------------------------' from dual
```

```
union all
select *
  from (select
        rpad('|'||substr(lpad(' ',1*(depth-1))||operation||
              decode(options, null,',' ||options), 1, 32), 33, ' ')||'|'||
        rpad(decode(id, 0, '----- '||to_char(hash_value)||' -----'
                         , substr(decode(substr(object_name, 1, 7), 'SYS_LE_', null, object_name)
                           ||' ',1, 20)), 21, ' ')||'|'||
        lpad(decode(cardinality,null,'    ',
                    decode(sign(cardinality-1000), -1, cardinality||' ',
                    decode(sign(cardinality-1000000), -1, trunc(cardinality/1000)||'K',
                    decode(sign(cardinality-1000000000), -1, trunc(cardinality/1000000)||'M',
                           trunc(cardinality/1000000000)||'G')))), 7, ' ') || '|' ||
        lpad(decode(bytes,null,' ',
                    decode(sign(bytes-1024), -1, bytes||' ',
                    decode(sign(bytes-1048576), -1, trunc(bytes/1024)||'K',
                    decode(sign(bytes-1073741824), -1, trunc(bytes/1048576)||'M',
                           trunc(bytes/1073741824)||'G')))), 6, ' ') || '|' ||
        lpad(decode(cost,null,' ',
                    decode(sign(cost-10000000), -1, cost||' ',
                    decode(sign(cost-1000000000), -1, trunc(cost/1000000)||'M',
                           trunc(cost/1000000000)||'G'))), 8, ' ') || '|' as "Explain plan"
            from v\$sql_plan where hash_value = $1
            and child_number = (select max(child_number) from v\$sql_plan where hash_value = $1))
union all
select '--------------------------------------------------------------------------------' from dual;

exit
EOF
```

以下介绍一个实际的诊断案例供参考。

曾经遇到过这样一次性能问题，开发人员编写的一个存储过程，其中包含了一系列的事务处理，大约有 10 个左右的 DML 事务执行，每一个 SQL 在 SQL*Plus 中执行都很迅速，但是一旦放在过程中执行，通过参数传入一个 sysdate，整个过程的执行就变得非常缓慢，无法成功完成。数据库环境为 Oralce9iR2：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.4.0 - 64bit Production
```

收到这个问题首先需要确定哪个 SQL 是真正引起性能问题的罪魁祸首。首先对 Procedure

进行一点修改，在每个 DML 事务执行之前在一张临时创建的测试表中插入一个时间，前后两个时间相减可以得到每个 SQL 独立执行的时间。通过这个办法，发现在执行到第 8 个 SQL 时，响应失去。也就是说，这个 SQL 是导致性能缓慢的根本原因。

找到这个 SQL 接下来的事情就变得简单一些，修改这个过程，在 Procedure 之前增加一个跟踪：

```
create or replace procedure cmop_servdetail_d_eygle (m_datstat date)    Authid Current_User is
  begin
  execute immediate 'alter session set sql_trace=true';
……
```

那么当再次执行这个过程时，SQL 的执过程被记录到一个 Trace 文件中，但是注意，由于 SQL 可能暂时无法完成，所以执行计划等信息并不会输出，但是由于 SQL 的 HASH 过程首先完成，在 Trace 文件的输出中，首先打印出了 SQL 的 HASH VALUE 值，这里是：hv=3740055767

```
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====================
PARSING IN CURSOR #2 len=32 dep=1 uid=28 oct=42 lid=28 tim=12567557125437 hv=3943786303 ad='20e289d8'
alter session set sql_trace=true
END OF STMT
EXEC #2:c=0,e=7735,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=12567557124267
=====================
PARSING IN CURSOR #2 len=4721 dep=1 uid=28 oct=2 lid=28 tim=12567557130962 hv=3740055767 ad='1270b8c0'
INSERT into cmo_servdetail_d
        (vc2dayguid,
。。。。。。。。
              and a.vc2bt = c.vc2cmbt
              and a.vc2cid = c.vc2cmcid
              and c.numsvrguid = b.numsvrguid
            group by b.numsvrguid,
                     b.vc2service_id,
                     substr(a.vc2mid, 0, 4),
                     a.vc2ua,
                     c.vc2urltype
END OF STMT
```

得到这个 HASH VALUE 值之后就可以通过 v$sql_plan 来获得这个 SQL 的执行计划，另外一个需要说明的是，我在诊断某个 SQL 问题时通常习惯将这个 SQL 创建到一张临时表中，以避免可能对 v$sql_plan 的反复查询带来的消耗。

对于这个案例我只需要发出如下语句：

```
create table t as select * from v$sql_plan where hash_value=3740055767
```

通过查询可以获得这条问题 SQL 的执行计划：

```
-------------------------------------------------------------------------------
| Operation                          | PHV/Object Name       |  Rows | Bytes|    Cost |
-------------------------------------------------------------------------------
|INSERT STATEMENT                    |----- 3740055767 ----|       |      |     123 |
|SORT GROUP BY                       |                       |     2 |  532 |     123 |
| FILTER                             |                       |       |      |         |
|  HASH JOIN                         |                       |     2 |  532 |     115 |
|   TABLE ACCESS BY LOCAL INDEX R|CM_URLLOG_JUMP            |     1 |  127 |     100 |
|    NESTED LOOPS                    |                       |     1 |  201 |     100 |
|     TABLE ACCESS FULL              |HY_SVR_REF_URL_CMCID |    11 |  814 |       3 |
|     PARTITION RANGE ITERATOR       |                       |       |      |         |
|      BITMAP CONVERSION TO ROWID|                         |       |      |         |
|       BITMAP AND                   |                       |       |      |         |
|        BITMAP CONVERSION FROM R|                         |       |      |         |
|         SORT ORDER BY              |                       |       |      |         |
|          INDEX RANGE SCAN          |CMIDX_URLLOG_JUMP_DA |     6 |      |       1 |
|        BITMAP CONVERSION FROM R|                         |       |      |         |
|         INDEX RANGE SCAN           |CMIDX_URLLOG_JUMP_CI |     6 |      |      19 |
|   TABLE ACCESS FULL                |HYO_SERVICEPLAN        |   10K|  694K|      14 |
-------------------------------------------------------------------------------
```

这就是这个 SQL 的执行计划，来对比一下 SQL*Plus 中执行这条 SQL 的执行计划：

```
Execution Plan
----------------------------------------------------------
   0      INSERT STATEMENT Optimizer=CHOOSE (Cost=35 Card=12 Bytes=3192)
   1    0   SORT (GROUP BY) (Cost=35 Card=12 Bytes=3192)
   2    1     FILTER
   3    2       HASH JOIN (Cost=26 Card=12 Bytes=3192)
   4    3         HASH JOIN (Cost=11 Card=1 Bytes=201)
   5    4           TABLE ACCESS (FULL) OF 'HY_SVR_REF_URL_CMCID' (Cost=3 Card=11 Bytes=814)
   6    4           PARTITION RANGE (ITERATOR)
   7    6             TABLE ACCESS (BY LOCAL INDEX ROWID) OF 'CM_URLLOG_JUMP' (Cost=6 Card=127720
Bytes=16220440)
   8    7               INDEX (RANGE SCAN) OF 'CMIDX_URLLOG_JUMP_DAT'(NON-UNIQUE) (Cost=2
Card=229897)
   9    3         TABLE ACCESS (FULL) OF 'HYO_SERVICEPLAN' (Cost=14 Card=10946 Bytes=711490)
```

注意到这两个执行计划完全不同，速度快的执行计划对于 PARTITION RANGE 访问是通过一个索引来完成的；而对于速度慢的执行计划，这里却使用了 2 个索引进行位图转换：

```
PARTITION RANGE                    ITERATOR
```

```
  BITMAP CONVERSION              TO ROWIDS
    BITMAP AND
      BITMAP CONVERSION          FROM ROWIDS
        SORT                     ORDER BY
          INDEX                                                    RANGE      SCAN
CMIDX_URLLOG_JUMP_DAT
      BITMAP CONVERSION          FROM ROWIDS
        INDEX                                                      RANGE      SCAN
CMIDX_URLLOG_JUMP_CID
```

正是这个转换使得性能大为缩减。

显然这个错误的选择是由于 CMIDX_URLLOG_JUMP_CID 索引的存在，再加上绑定变量的影响，CBO 最终选择了错误的执行计划，为了快速的解决问题，在确认之后，我们直接 Drop 掉了这个索引。

此时再次运行过程，发现很快完成，此时的执行计划通过同样的方法可以获得：

```
-------------------------------------------------------------------------------
| Operation                          | PHV/Object Name        |   Rows | Bytes|     Cost |
-------------------------------------------------------------------------------
|INSERT STATEMENT                    |----- 3740055767 ----|          |         |       35 |
|SORT GROUP BY                       |                              |      12 |      3K|       35 |
| FILTER                             |                              |         |        |          |
|   HASH JOIN                        |                              |      12 |      3K|       26 |
|    HASH JOIN                       |                              |       1 |   201 |       11 |
|     TABLE ACCESS FULL              |HY_SVR_REF_URL_CMCID |      11 |   814 |        3 |
|     PARTITION RANGE ITERATOR       |                              |         |        |          |
|      TABLE ACCESS BY LOCAL INDEX|CM_URLLOG_JUMP          |    128K|    15M|        6 |
|       INDEX RANGE SCAN             |CMIDX_URLLOG_JUMP_DA |    230K|        |        2 |
|    TABLE ACCESS FULL               |HYO_SERVICEPLAN         |     10K|   694K|       14 |
-------------------------------------------------------------------------------
```

现在的执行计划恢复了正常。注意到错误的执行计划选择了 bitmap convert 的执行计划，而两个索引都是 B*Tree 索引。这种转换是 Oracle9i 引入的，同时一个隐含参数被用来控制这种转换：

```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.ksppdesc describ
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3   WHERE x.inst_id = USERENV ('Instance')
  4     AND y.inst_id = USERENV ('Instance')
  5     AND x.indx = y.indx
  6     AND x.ksppinm LIKE '%&par%'
  7  /
Enter value for par: _b_tree_bitmap_plans
```

```
old   6:      AND x.ksppinm LIKE '%&par%'
new   6:      AND x.ksppinm LIKE '%_b_tree_bitmap_plans%'
NAME                                VALUE    DESCRIB
------------------------- ------ ------------------------------------
_b_tree_bitmap_plans        TRUE    enable the use of bitmap plans for tables w. only B-tree indexes
```

如果这种性能衰减的转换经常发生，可以将这个隐含参数设置为 FALSE.

9.2.2 EXPLAIN PLAN FOR 与 DBMS_XPLAN

在前面已经简单提到过，Explain Plan For 和 DBMS_XPLAN 包结合可以用于获取 SQL 的执行计划。本节我们将对这两者的结合使用进行进一步的介绍。

EXPLAIN PLAN 命令可以在后台对 SQL 进行解析，并将 SQL 执行计划加载到执行计划表中（默认名称为 PLAN_TABLE），这是 EXPLAIN PLAN 的作用，其通常的使用方法是在 SQL*PLUS 中输入类似如下命令：

Explain plan <set statement_id = 'text'> <into your plan table>for statement

其中通过 set statement_id = 'text' 可以为 SQL 进行名称标记，"into your plan table"默认的是 plan_table 表，通常使用格式如下：

```
EXPLAIN PLAN FOR
SELECT * FROM emp WHERE empno=7788;
```

执行计划生成之后，剩下的就是展现问题，DBMS_XPLAN 包就是用来实现这一功能的，该 PACKAGE 自 Oracle9iR2 引入，初始的只具有一个函数：

```
SQL> desc dbms_xplan
FUNCTION DISPLAY RETURNS DBMS_XPLAN_TYPE_TABLE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 TABLE_NAME                     VARCHAR2                IN     DEFAULT
 STATEMENT_ID                   VARCHAR2                IN     DEFAULT
 FORMAT                         VARCHAR2                IN     DEFAULT
```

而在 Oracle10g 中该 Package 的功能得到了极大的增强。

DBMS_XPLAN.DISPLAY 可以被调用来返回执行计划，调用过程类似如下：

*SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);*

值得注意的是，DBMS_XPLAN 还能从存储 SGA 内的指针显示"实时"执行计划，通过查看 V$SESSION 视图，可以找到会话执行 SQL 的 SQL ID，拥有了这个 SQL ID 之后可以通过 DBMS_XPLAN.DISPLAY_CURSOR 来获得 Cursor 所使用的执行计划。

此外 DBMS_XPLAN.DISPLAY_AWR 函数还可用来查询 Oracle 10g 的自动工作负载库(Automatic Workload Repository, AWR)获得的历史 SQL 语句，并显示它的执行计划。

下面我们通过 Oracle10g 中的测试应用来展示一下这个 Package 对于 SQL 执行计划的获取与展现。

首先通过 EXPLAIN PLAN 来生成执行计划：

```
SQL> EXPLAIN PLAN set statement_id = 'NO' FOR
  2  SELECT * FROM emp WHERE empno=7788;
Explained.
```

然后通过查询展现以上生成的执行计划：

```
SQL> SELECT plan_table_output
  2  FROM     TABLE( DBMS_XPLAN.DISPLAY('PLAN_TABLE','NO','ALL') );
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2949544139
--------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    87 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    87 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PK_EMP  |     1 |       |     1   (0)| 00:00:01 |
--------------------------------------------------------------------------------
Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
   1 - SEL$1 / EMP@SEL$1
   2 - SEL$1 / EMP@SEL$1
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPNO"=7788)
Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "EMPNO"[NUMBER,22], "EMP"."ENAME"[VARCHAR2,10],
       "EMP"."JOB"[VARCHAR2,9], "EMP"."MGR"[NUMBER,22], "EMP"."HIREDATE"[DATE,7],
       "EMP"."SAL"[NUMBER,22], "EMP"."COMM"[NUMBER,22], "EMP"."DEPTNO"[NUMBER,22]
   2 - "EMP".ROWID[ROWID,10], "EMPNO"[NUMBER,22]
28 rows selected.
```

以上功能可以用于显示已知 SQL 的执行计划，但是很多时候我们需要直接从数据库中获得其他应用 SQL 的执行计划。从 Oracle9i 开始，V$SQL_PLAN_STATISTICS、V$SQL_PLAN 视图被引入用于记录 SQL 的执行统计信息以及执行计划，但是在 Oracle9i 中，从以上视图获取执行计划通常需要自己手工编写脚本，实现起来较为复杂，在 Oracle10g 中新的增强被引入，DBMS_XPLAN.DISPLAY_CURSOR 可以很容易的帮助我们实现以上需求，执行该功能需要对 V$SESSION、V$SQL、V$SQL_PLAN 、V$SQL_PLAN_STATISTICS_ALL 就有访问权限。此时使用 scott 用户进行，需要首先对 SCOTT 用户授权：

```
grant select on v_$session to scott;
grant select on v_$sql_plan to scott;
```

```
grant select on v_$sql to scott;
```

DISPLAY_CURSOR 的参数需要如下：

```
FUNCTION DISPLAY_CURSOR RETURNS DBMS_XPLAN_TYPE_TABLE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SQL_ID                         VARCHAR2                IN     DEFAULT
 CURSOR_CHILD_NO                NUMBER(38)              IN     DEFAULT
 FORMAT                         VARCHAR2                IN     DEFAULT
```

现在看看 DISPLAY_CURSOR 的输出：

```
SQL> SELECT plan_table_output
  2  FROM   TABLE(DBMS_XPLAN.DISPLAY_CURSOR);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID  1m225m1612xvg, child number 0
-------------------------------------
SELECT d.dname, SUM(e.sal) AS sum_sal FROM   dept d,emp  e WHERE
d.deptno = e.deptno GROUP  BY  d.dname
Plan hash value: 2006461124
--------------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |       |       |     8 (100)|          |
|   1 |  HASH GROUP BY               |      |    14 |   672 |     8  (25)| 00:00:01 |
|*  2 |   HASH JOIN                  |      |    14 |   672 |     7  (15)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| DEPT    |     4 |    88 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| EMP     |    14 |   364 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("D"."DEPTNO"="E"."DEPTNO")
Note
-----
   - dynamic sampling used for this statement
```

在 Oracle10g 中，可以通过 V$SESSION 或 V$SQL 等视图来获取不同会话的 SQL_ID 以及 SQL_CHILD_NUMBER 来获得其 SQL 执行计划：

```
SQL> select sid,username,sql_id,sql_child_number
  2  from v$session where sql_id is not null;
       SID USERNAME                       SQL_ID        SQL_CHILD_NUMBER
---------- ------------------------------ ------------- ----------------
```

```
         360 SMSNP                      bgg2p17bx7y1v              2
         372 SYS                        6zn53xvm7zth4              0
         386 SMSNP                      1ug8zqhw906tb              0
         393 SMSNP                      fx3kcsrcm7hcb            106
         396 SMSNP                      4t47xyz571hrh             90
         401 SMSNP                      1ug8zqhw906tb              0
         405 SYS                        65vuzhm491wk9              1
         422 SMSNP                      1bkv8cyq1c9f1              1
         429                            4gd6b1r53yt88              0
```

注意，某些 SQL 的执行计划可能老化无法获取，以下是通过 v$session 信息获得的一个 SQL 执行计划：

```
SQL> SELECT plan_table_output
  2  FROM     TABLE(DBMS_XPLAN.DISPLAY_CURSOR('bgg2p17bx7y1v',2,'ALL'));
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID  bgg2p17bx7y1v, child number 2
-------------------------------------
SELECT COUNT(*) FROM "NP_IO_MT" "A1" WHERE
"A1"."MSGDATE">SYSDATE@!-.0104166666666666666666666666666666666667 AND
"A1"."HYSERVICECODE"='QWOZQX' AND  EXISTS (SELECT 0 FROM "NP_IO_MO" "A2" WHERE
"A2"."PHONE"="A1"."PHONE" AND
"A2"."MSGDATE">SYSDATE@!-.0104166666666666666666666666666666666667 AND
"A2"."HYSERVICECODE"='HZXDBM')
Plan hash value: 4119176645
-----------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name                | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                     |       |       |     9 (100)|          |       |       |
|   1 |  SORT AGGREGATE                    |                     |     1 |    56 |            |          |       |       |
|   2 |   NESTED LOOPS SEMI                |                     |     1 |    56 |     9   (0)| 00:00:01 |       |       |
|   3 |    PARTITION RANGE ITERATOR        |                     |     1 |    29 |     5   (0)| 00:00:01 |   KEY |    19 |
|*  4 |     TABLE ACCESS BY LOCAL INDEX ROWID| NP_IO_MT          |     1 |    29 |     5   (0)| 00:00:01 |   KEY |    19 |
|*  5 |      INDEX RANGE SCAN              | IND_IO_MT_MSGDATE   |     6 |       |     3   (0)| 00:00:01 |   KEY |    19 |
|   6 |    PARTITION RANGE ITERATOR        |                     |     1 |    27 |     4   (0)| 00:00:01 |   KEY |    19 |
|*  7 |     TABLE ACCESS BY LOCAL INDEX ROWID| NP_IO_MO          |     1 |    27 |     4   (0)| 00:00:01 |   KEY |    19 |
|*  8 |      INDEX RANGE SCAN              | IND_IO_MO_MSGDATE   |     9 |       |     2   (0)| 00:00:01 |   KEY |    19 |
-----------------------------------------------------------------------------------------------------
Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
```

```
   1 - SEL$5DA710D3
   4 - SEL$5DA710D3 / A1@SEL$1
   5 - SEL$5DA710D3 / A1@SEL$1
   7 - SEL$5DA710D3 / A2@SEL$2
   8 - SEL$5DA710D3 / A2@SEL$2
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("A1"."HYSERVICECODE"='QWOZQX')
   5 - access("A1"."MSGDATE">SYSDATE@!-.010416666666666666666666666666666666666667)
   7 - filter(("A2"."HYSERVICECODE"='HZXDBM' AND "A2"."PHONE"="A1"."PHONE"))
   8 - access("A2"."MSGDATE">SYSDATE@!-.010416666666666666666666666666666666666667)
Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - (#keys=0) COUNT(*)[22]
   3 - "A1"."PHONE"[VARCHAR2,40]
   4 - "A1"."PHONE"[VARCHAR2,40]
   5 - "A1".ROWID[ROWID,10]
   8 - "A2".ROWID[ROWID,10]
```

9.2.3 通过 AWR 获取 SQL 执行计划

前面介绍的 DBMS_XPLAN 包还有另外一个功能，通过 dbms_xplan.display_awr 函数来获取 AWR 中的 SQL 执行计划。这个函数的主要参数需要是 SQL_ID:

```
FUNCTION DISPLAY_AWR RETURNS DBMS_XPLAN_TYPE_TABLE
参数名称                       类型                    输入/输出默认值?
------------------------------ ----------------------- ------ --------
SQL_ID                         VARCHAR2                IN
PLAN_HASH_VALUE                NUMBER(38)              IN     DEFAULT
DB_ID                          NUMBER(38)              IN     DEFAULT
FORMAT                         VARCHAR2                IN     DEFAULT
```

通过 AWR 生成的报告关于 SQL 部分都会包含 SQL_ID 一项内容，通过这个 SQL_ID 就可以查询 AWR 中的 SQL 执行计划，以下是 Oracle10g　AWR 报告的一个输出片段：

```
  Elapsed      CPU                  Elap per  % Total
  Time (s)   Time (s)  Executions  Exec (s)  DB Time    SQL Id
---------- ---------- ------------ ---------- ------- -------------
         9          2        1,083        0.0     6.9 1dbqwa8xts1g3
insert into HY_SIMULATESRC (DATTIMESTAMP, VC2SRCDESC, VC2MOBILE, VC2IMSI, VC2MOB
ILEIP, VC2MEMO, VC2PLANKEY, VC2SRCTYPE, VC2UA, VC2PROVID, NUMGUID) values (:1, :
2, :3, :4, :5, :6, :7, :8, :9, :10, :11)
```

获得 SQL 的 SQL_ID，就可以通过 DBMS_XPLAN 来输出执行计划，以下输出的 SQL 具有多个子指针，执行计划各不相同，在使用绑定变量的情况下，Oracle 数据库也会通过绑定变量 Peeking 来获得更为准确的执行：

```
SQL> select * from table(dbms_xplan.display_awr('072t81cu41xfj'));
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID 072t81cu41xfj
--------------------
SELECT DECODE(COUNT(*), 0, 1, 0) FROM MGMT_SEVERITY WHERE TARGET_GUID = :B3 AND
METRIC_GUID = :B2 AND KEY_VALUE = :B1
Plan hash value: 356059170
--------------------------------------------------------------------------------
| Id  | Operation              | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                            |       |       |     1 (100)|          |
|   1 |  SORT AGGREGATE        |                            |     1 |    41 |            |          |
|   2 |   INDEX RANGE SCAN| SEVERITY_PRIMARY_KEY |     1 |    41 |     1   (0)| 00:00:01 |
--------------------------------------------------------------------------------

SQL_ID 072t81cu41xfj
--------------------
SELECT DECODE(COUNT(*), 0, 1, 0) FROM MGMT_SEVERITY WHERE TARGET_GUID = :B3 AND
METRIC_GUID = :B2 AND KEY_VALUE = :B1
Plan hash value: 2975161209
--------------------------------------------------------------------------------
| Id  | Operation              | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                            |       |       |    24 (100)|          |
|   1 |  SORT AGGREGATE        |                            |     1 |    40 |            |          |
|   2 |   INDEX FAST FULL SCAN | SEVERITY_PRIMARY_KEY |  1966 | 78640 |    24   (0)|
00:00:01|
--------------------------------------------------------------------------------
已选择 30 行。
```

通过 dbms_xplan.display_awr 函数获取的 SQL 执行计划来自 dba_hist_sql_plan 视图，通过历史数据记录，甚至一些被老化的 SQL 执行计划仍然可以被查询到。

9.3捕获问题 SQL 解决过度 CPU 消耗问题

在生产环境中，我们可能会经常遇到 CPU 过度使用而影响系统性能或正常运行的问题。大多数情况下，系统的性能问题都是由不良 SQL 代码引起的，那么作为 DBA，怎样发现和解决这些 SQL 问题就显得尤为重要。

本案例的系统环境为:

操作系统: Solaris8　　数据库版本: 8.1.7.4

问题描述:业务及开发人员报告系统运行缓慢，已经影响业务系统正常使用。请求协助诊断。

9.2.4 使用 vmstat 检查系统当前情况

首先登陆数据库主机，检查当前系统状况。

使用 vmstat 检查，发现 CPU 资源已经耗尽，大量任务位于运行队列：

```
bash-2.03$ vmstat 3
 procs       memory                  page                    disk          faults          cpu
 r b w      swap    free    re   mf pi po fr de sr s6 s9 s1 sd     in      sy      cs us sy id
 0 0 0 5504232 1464112 0    0    0    0    0    0    0    0    1    1    0 4294967196 0 0 -84 -5 -145
 131 0 0 5368072 1518360 56 691 0 2 2 0    0    0    1    0    0 3011 7918 2795 97    3    0
 131 0 0 5377328 1522464 81 719 0 2 2 0    0    0    1    0    0 2766 8019 2577 96    4    0
 130 0 0 5382400 1524776 67 682 0 0 0 0    0    0    0    0    0 3570 8534 3316 97    3    0
 134 0 0 5373616 1520512 127 1078 0 2 2 0 0 0    1    0    0 3838 9584 3623 96    4    0
 136 0 0 5369392 1518496 107 924 0 5 5 0 0    0    0    0    0 2920 8573 2639 97    3    0
 132 0 0 5364912 1516224 63 578 0 0 0 0    0    0    0    0    0 3358 7944 3119 97    3    0
 129 0 0 5358648 1511712 189 1236 0 0 0 0 0 0 0    0    0    0 3366 10365 3135 95 5    0
 129 0 0 5354528 1511304 120 1194 0 0 0 0 0 0 0    0    4    0 3235 8864 2911 96    4    0
```

对于 vmstat 的用法及输出，简要说明一下：vmstat 是 Unix 上一个常用的工具，可以帮助我们查看系统内存及 CPU 使用情况。

Vmstat 最常用的两个参数是　t [n]:

t　表示采样间隔

n　表示采样次数

例如:vmstat 5 5,表示在 T(5)秒时间内进行 N(5)次采样。

对于前三列 procs 输出，分别代表以下含义:

r-->指运行队列中的进程数，如果这个参数经常超过 CPU 数量可能说明 CPU 存在瓶颈

b-->因为 IO 被 Block 的进程数

w-->idle 的被 swap 的进程数

最后一项 CPU 标识系统 CPU 资源的分配和使用情况，最后一列 Idle 值通常被用来衡量系统 CPU 的空闲情况。

本案例当时，系统 CPU 资源已经耗尽，Idle 为 0，并且运行队列大量进程排队等待。

9.2.5 使用 Top 工具辅助诊断

通过 Top 工具，可以查看进程 CPU 耗用情况，如果存在进程异常，可以通过 Top 定位，为进一步诊断提供依据。对于本案例，观察进程 CPU 耗用，发现没有明显过高 CPU 使用的进程。

```
$ top

last pid: 28313;    load averages: 99.90, 117.54, 125.71                    23:28:38
296 processes: 186 sleeping, 99 running, 2 zombie, 9 on cpu
CPU states:    0.0% idle, 96.5% user,    3.5% kernel,    0.0% iowait,    0.0% swap
Memory: 4096M real, 1404M free, 2185M swap in use, 5114M swap free

   PID USERNAME THR PRI NICE    SIZE      RES STATE       TIME       CPU COMMAND
 27082 oracle8i     1    33        0 1328M 1309M run           0:17    1.29% oracle
 26719 oracle8i     1    55        0 1327M 1306M sleep         0:29    1.11% oracle
 28103 oracle8i     1    35        0 1327M 1304M run           0:06    1.10% oracle
 28161 oracle8i     1    25        0 1327M 1305M run           0:04    1.10% oracle
 26199 oracle8i     1    45        0 1328M 1309M run           0:42    1.10% oracle
 26892 oracle8i     1    33        0 1328M 1310M run           0:24    1.09% oracle
 27805 oracle8i     1    45        0 1327M 1306M cpu/1         0:10    1.04% oracle
 23800 oracle8i     1    23        0 1327M 1306M run           1:28    1.03% oracle
 25197 oracle8i     1    34        0 1328M 1309M run           0:57    1.03% oracle
```

从 Top 的输出中我们发现有大量进程处于 running 的运行状态，CPU 消耗很平均，单进程消耗大约在 1%左右，基本可以排除个别进程异常导致 CPU 问题的可能（关于单进程异常 CPU 消耗问题可以参考第四章中的解决方法）。

9.2.6 检查进程数量

对于一个生产数据库系统，稳定运行的进程数量通常是可知的。

> 提示:对于稳定运行的生产系统，数据库的运行状况通常是稳定的，如果你绘制出性能曲线，你会发现每个星期的曲线几乎是可以重合的，对数据库系统的运行状况及性能指标具有充分认识和了解是必须的。

看一下当前系统的进程数量，从而进行比较判断：

```
bash-2.03$ ps -ef|grep ora|wc -l
       258
bash-2.03$ ps -ef|grep ora|wc -l
       275
bash-2.03$ ps -ef|grep ora|wc -l
       274
bash-2.03$ ps -ef|grep ora|wc -l
       278
bash-2.03$ ps -ef|grep ora|wc -l
       277
bash-2.03$ ps -ef|grep ora|wc -l
       366
```

发现此时系统存在大量 Oracle 进程,大约在 300 左右，大量进程消耗了几乎所有 CPU 资源,而正常情况下 Oracle 连接数应该在 100 左右。由此，可以做出基本判断，是数据库或应用出现问题，导致进程任务无法完成，不断累积，从而出现大量队列等待。

这些等待在数据库中应该有具体的体现，接下来需要登陆数据库进行检查了。

9.2.7 登陆数据库

我们判断数据库可能经历了等待，那么 Oracle 数据库提供了相关视图供我们查询和发现问题，v$session_wait 是首先值得我们关注的。查询 v$session_wait 获取各进程等待事件：

```
SQL> select sid,event,p1,p1text from v$session_wait;
       SID EVENT                                          P1 P1TEXT
---------- ------------------------------ ---------- ------------------------------
       124 latch free                        1.6144E+10 address
         1 pmon timer                               300 duration
         2 rdbms ipc message                        300 timeout
      。。。。。。
       140 buffer busy waits                         17 file#
        66 buffer busy waits                         17 file#
        10 db file sequential read                   17 file#
        18 db file sequential read                   17 file#
        54 db file sequential read                   17 file#
        49 db file sequential read                   17 file#
        48 db file sequential read                   17 file#
```

```
          46 db file sequential read                  17 file#
          45 db file sequential read                  17 file#
。。。。。。
         234 db file sequential read                  17 file#
         233 db file sequential read                  17 file#
         230 db file sequential read                  17 file#
         333 db file sequential read                  17 file#
         330 db file scattered read                   17 file#
         310 db file scattered read                   17 file#
。。。。。。。

244 rows selected.
```

对于本案例，我们发现存在大量 db file scattered read 及 db file sequential read 等待。显然全表扫描等操作成为系统最严重的性能影响因素。

9.2.8 捕获相关 SQL

确定这些进程因为数据访问产生了等待，我们考虑捕获这些 SQL 以发现问题。

这里用到了我的以下脚本 getsqlbysid.sql，该脚本通过已知 session 的 sid，联合 v$session、v$sqltext 视图，获得相关 session 正在执行的完整的 SQL 语句。

```
SELECT     sql_text
      FROM v$sqltext a
     WHERE a.hash_value = (SELECT sql_hash_value
                                    FROM v$session b
                                   WHERE b.SID = '&sid')
ORDER BY piece ASC
/
```

使用该脚本,通过从 v$session_wait 中获得的等待全表或索引扫描的进程 SID，可以捕获可能存在问题的 sql 语句:

```
SQL> @getsqlbysid
Enter value for sid: 18
old      5: where b.sid='&sid'
new      5: where b.sid='18'

SQL_TEXT
------------------------------------------------------------------
select i.vc2title,i.numinfoguid    from    hs_info i where i.intenab
ledflag = 1    and i.intpublishstate = 1    and i.datpublishdate <=
sysdate    and i.numcatalogguid = 2047 order by i.datpublishdate d
```

```
esc, i.numorder desc
```

对几个进程进行跟踪,分别得到以上 SQL 语句，这些 SQL 可能就是问题产生的根源（以上语句如果良好编码应该使用绑定变量，但是目前这个不是我们关心的）。

使用该应用用户连接，通过 autotrace 功能检查以上 SQL 的执行计划:

```
SQL> set autotrace trace explain
SQL> select i.vc2title,i.numinfoguid
  2  from    hs_info i where i.intenabledflag = 1
  3  and i.intpublishstate = 1    and i.datpublishdate <=sysdate
  4  and i.numcatalogguid = 3475
  5  order by i.datpublishdate desc, i.numorder desc    ;
Execution Plan
----------------------------------------------------------
   0        SELECT STATEMENT Optimizer=CHOOSE (Cost=228 Card=1 Bytes=106)
   1    0     SORT (ORDER BY) (Cost=228 Card=1 Bytes=106)
   2    1       TABLE ACCESS (FULL) OF 'HS_INFO' (Cost=218 Card=1 Bytes=106)
SQL> select count(*) from hs_info;
  COUNT(*)
----------
    227404
```

通过执行计划，我们看到以上查询使用了全表扫描，而该表这里有 22 万记录，全表扫描已经不再适合。通常对于小表，Oracle 建议通过全表扫描进行数据访问，对于大表则应该通过索引以加快数据查询，当然如果查询要求返回表中大部分或者全部数据，那么全表扫描可能仍然是最好的选择。

从 V$SYSSTAT 视图中，我们可以查询得到关于全表扫描的系统统计信息：

```
SQL> col name for a30
SQL> select name,value from v$sysstat
  2  where name in ('table scans (short tables)','table scans (long tables)');

NAME                           VALUE
------------------------------ ----------
table scans (short tables)      828
table scans (long tables)       101
```

其中 table scans (short tables)指对于小表的全表扫描的此时；table scans (long tables)指对于大表的全表扫描的次数。从 Statspack 的报告中，我们也可以找到这部分信息：

```
Instance Activity Stats for DB: CELLSTAR  Instance: ora8i  Snaps:        20 -
Statistic                                     Total     per Second     per Trans
--------------------------------- ---------------- ------------ ------------
。。。。。。
table scan blocks gotten                38,228,349           37.0           26.9
```

table scan rows gotten	546,452,583	528.9	383.8
table scans (direct read)	5,784	0.0	0.0
table scans (long tables)	5,990	0.0	0.0
table scans (rowid ranges)	5,850	0.0	0.0
table scans (short tables)	1,185,275	1.2	0.8

通常，如果一个数据库的 table scans (long tables)过多，那么 db file scattered read 等待事件可能同样非常显著，和以上数据来自同一个 report 的 Top5 等待事件就是如此：

Top 5 Wait Events

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

Event	Waits	Wait Time (cs)	% Total Wt Time
log file parallel write	1,436,993	1,102,188	10.80
log buffer space	16,698	873,203	8.56
log file sync	1,413,374	654,587	6.42
control file parallel write	329,777	510,078	5.00
db file scattered read	**425,578**	**132,537**	**1.30**

数据库内部，很多信息和现象都是紧密相关的，只要我们加深对于数据库的了解，在优化和诊断数据库问题时就能够得心应手。

Oracle 通过一个内部参数_small_table_threshold 来定义大表和小表的界限。缺省的该参数等于 2%的 Buffer 数量，如果表的大小小于该参数定义，Oracle 认为该表为小表，否则 Oracle 认为该表为大表。我们看一下 Oracle9iR2 中的情况：

```
SQL> @GetParDescrb.sql
Enter value for par: small
old     6:       AND x.ksppinm LIKE '%&par%'
new     6:       AND x.ksppinm LIKE '%small%'
NAME                                     VALUE          DESCRIB
-------------------------------- ---------- ---------------------------------
_small_table_threshold                 200              threshold level of table size for direct reads
```

以上数据库中，200 正好约为 Buffer 数量的 2%:

```
SQL> show parameter db_cache_size
NAME                                     TYPE             VALUE
------------------------------------ ----------- ------------------------------
db_cache_size                                    big integer 83886080
SQL> select (83886080/8192)*2/100 from dual;
(83886080/8192)*2/100
---------------------
                204.8
```

所以要区分大小表（Long/Short）是因为全表扫描可能引起 Buffer Cache 的抖动，缺省的大表的全表扫描会被置于 LRU 的末端，以期尽快老化，减少 Buffer 的占用。从 Oracle8i 开始，
~~~~~~~~~~~~~~~~~~

Oracle 的多缓冲池管理技术（Default/Keep/Recycle 池）给了我们另外一个选择，对于不同大小、不同使用频率的数据表，从建表之初就可以指定其存储 Buffer，以使得内存使用更加有效。

让我们继续以上的案例，在实际处理中，我们检查全表扫描的数据表，发现存在以下索引:

```
SQL> select index_name,index_type    from user_indexes where table_name='HS_INFO';

INDEX_NAME                      INDEX_TYPE
------------------------------ ---------------------------
HSIDX_INFO1                     FUNCTION-BASED NORMAL
HSIDX_INFO_SEARCHKEY            DOMAIN
PK_HS_INFO                      NORMAL
```

检查索引键值：

```
SQL> select index_name,column_name
     2 from user_ind_columns    where table_name ='HS_INFO';
INDEX_NAME                      COLUMN_NAME
------------------------------ --------------------
HSIDX_INFO1                     NUMORDER
HSIDX_INFO1                     SYS_NC00024$
HSIDX_INFO_SEARCHKEY            VC2INDEXWORDS
PK_HS_INFO                      NUMINFOGUID
SQL> desc hs_info
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------------
 NUMINFOGUID                               NOT NULL NUMBER(15)
 NUMCATALOGGUID                            NOT NULL NUMBER(15)
 INTTEXTTYPE                               NOT NULL NUMBER(38)
 VC2TITLE                                  NOT NULL VARCHAR2(60)
 VC2AUTHOR                                          VARCHAR2(100)
 NUMPREVINFOGUID                                    NUMBER(15)
 NUMNEXTINFOGUID                                    NUMBER(15)
 NUMORDER                                  NOT NULL NUMBER(15)
。。。。。
```

9.2.9 创建新的索引以消除全表扫描

检查发现在 numcatalogguid 字段上并没有索引,该字段具有很好的区分度，考虑在该字段创建索引以消除全表扫描。

```
SQL> create index hs_info_NUMCATALOGGUID on hs_info(NUMCATALOGGUID);
Index created.
```

```
SQL> set autotrace trace explain
SQL> select i.vc2title,i.numinfoguid
  2  from    hs_info i where i.intenabledflag = 1
  3  and i.intpublishstate = 1    and i.datpublishdate <=sysdate
  4  and i.numcatalogguid = 3475
  5  order by i.datpublishdate desc, i.numorder desc ;

Execution Plan
----------------------------------------------------------
   0        SELECT STATEMENT Optimizer=CHOOSE (Cost=12 Card=1 Bytes=106)
   1    0     SORT (ORDER BY) (Cost=12 Card=1 Bytes=106)
   2    1       TABLE ACCESS (BY INDEX ROWID) OF 'HS_INFO' (Cost=2 Card=1 Bytes=106)
   3    2         INDEX (RANGE SCAN) OF 'HS_INFO_NUMCATALOGGUID' (NON-UNIQUE)
(Cost=1 Card=1)
```

9.2.10观察系统状况

原大量等待消失

```
SQL> select sid,event,p1,p1text from v$session_wait where event not like 'SQL%';

       SID EVENT                                P1 P1TEXT
---------- ---------------------------------- ---------- ----------------------------------------------------------------
         1 pmon timer                                300 duration
         2 rdbms ipc message                         300 timeout
         3 rdbms ipc message                         300 timeout
         6 rdbms ipc message                      180000 timeout
        59 rdbms ipc message                        6000 timeout
       118 rdbms ipc message                        6000 timeout
       275 rdbms ipc message                       30000 timeout
       147 rdbms ipc message                        6000 timeout
        62 rdbms ipc message                        6000 timeout
        11 rdbms ipc message                       30000 timeout
         4 rdbms ipc message                         300 timeout
       305 db file sequential read                17 file#
       356 db file sequential read                17 file#
        19 db file scattered read                 17 file#
         5 smon timer                                300 sleep time

15 rows selected.
```

在另外的 session 里，持续观察的 CPU 使用情况：

```
bash-2.03$ vmstat 3
 procs      memory            page            disk          faults      cpu
 r b w    swap   free   re   mf pi po fr de sr s6 s9 s1 sd   in   sy   cs us sy id
 34 0 0 5343016 1465416 44 386 77 0 0 0   0   0   0   0   0 3197 8486 2902 92   8   0
 31 0 0 5331568 1459696 178 1491 122 0 0 0 0 0 0   3   0 3237 9461 3005 89 11   0
 31 0 0 5317792 1453008 76 719 80 0 0 0   0   0   0   0   0 3292 8736 3025 93   7   0
 31 2 0 5311144 1449552 235 1263 69 2 2 0 0 0   1   0   0 3473 9535 3357 88 12   0
 25 0 0 5300240 1443920 108 757 18 2 2 0 0   0   1   1   0 2377 7876 2274 95   5   0
 19 0 0 5295904 1441840 50 377 0 0 0 0   0   0   0   0   1   0 1915 6598 1599 98   1   0
----以上为创建索引之前部分
----以下为创建索引之后部分，CPU 使用率恢复正常
 procs      memory            page            disk          faults      cpu
 r b w    swap   free   re   mf pi po fr de sr s6 s9 s1 sd   in   sy   cs us sy id
0 0 0 4955872 1287136 737 6258 16 0 0 0 0   0   0   3   0 2890 11777 4432 44 12 44
 1 0 0 4887888 1256464 809 6234 8 2 2 0   0   0   0   2   0 2809 12066 4247 45 12 43
 0 0 0 4828912 1228200 312 2364 13 5 5 0 0   0   2   1   0 2410 6816 3492 38   6 57
 0 0 0 4856816 1240168 8 138 0   0   0   0   0   0   1   0   0 2314 4026 3232 34   4 62
 0 0 0 4874176 1247712 0 86   0   0   0   0   0   0   0   0   0 2298 3930 3324 35   2 63
 2 0 0 4926088 1270824 34 560 0 0   0   0   0   0   0   0   0 2192 4694 2612 29 16 55
 0 0 0 5427320 1512952 53 694 0 0   0   0   0   0   3   2   0 2443 5085 3340 33 12 55
 0 0 0 5509120 1553136 0 37   0   0   0   0   0   0   0   0   0 2309 3908 3321 35   1 64
```

至此，此问题得以解决。

9.2.11性能何以提高

回答这个问题似乎是多余的,我只想重申一点:

有效的降低 SQL 的逻辑读是 SQL 优化的基本原则之一。

我们来比较一下前后两种执行方式的逻辑读取及性能差异。

a. 全表扫描的性能

```
SQL> select i.vc2title,i.numinfoguid
  2  from   hs_info i where i.intenabledflag = 1
  3  and i.intpublishstate = 1   and i.datpublishdate <=sysdate
  4  and i.numcatalogguid = 3475
  5  order by i.datpublishdate desc, i.numorder desc   ;
352 rows selected.
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=541 Card=1 Bytes=106)
```

```
   1      0    SORT (ORDER BY) (Cost=541 Card=1 Bytes=106)
   2      1      TABLE ACCESS (FULL) OF 'HS_INFO' (Cost=531 Card=1 Bytes=106)
Statistics
----------------------------------------------------------
          0   recursive calls
         25   db block gets
       3499   consistent gets
        258   physical reads
          0   redo size
…..
          2   sorts (memory)
          0   sorts (disk)
        352   rows processed
```

b.　使用索引的性能

```
SQL> select i.vc2title,i.numinfoguid
  2  from   hs_info i where i.intenabledflag = 1
  3  and i.intpublishstate = 1    and i.datpublishdate <=sysdate
  4  and i.numcatalogguid = 3475
  5  order by i.datpublishdate desc, i.numorder desc;
352 rows selected.

Execution Plan
----------------------------------------------------------
   0        SELECT STATEMENT Optimizer=CHOOSE (Cost=12 Card=1 Bytes=106)
   1      0    SORT (ORDER BY) (Cost=12 Card=1 Bytes=106)
   2      1      TABLE ACCESS (BY INDEX ROWID) OF 'HS_INFO' (Cost=2 Card=1 Bytes=106)
   3      2        INDEX (RANGE SCAN) OF 'HS_INFO_NUMCATALOGGUID' (NON-UNIQUE)
(Cost=1 Card=1)
Statistics
----------------------------------------------------------
          0   recursive calls
          0   db block gets
         89   consistent gets
          0   physical reads
          0   redo size
….
          1   sorts (memory)
          0   sorts (disk)
        352   rows processed
```

consistent gets 从 **3499** 到 **89**,我们看到性能得到了巨大的提高.

结语:

通常,开发人员很少注意 SQL 代码的效率,他们更着眼于功能的实现。

至于性能问题通常被认为是次要的,而且在应用系统开发初期,由于数据库数据量较少,对于查询 SQL 语句等,不容易体会出各种 SQL 句法的性能差异。

但是一旦这些应用作为生产系统上线运行,随着数据库中数据量的增加,大量并发访问,系统的响应速度可能就会成为系统需要解决的最主要的问题之一。

在少量用户下性能可以接受的 SQL，可能在大量用户并发的条件下就会成为性能瓶颈。

在我这个案例中，开发人员很难相信仅只一条 SQL 语句就导致了整个数据库的性能下降。然而事实就是如此，一条低效的 SQL 语句就可能毁掉你的数据库，所以在系统设计及开发过程中，你必须考虑到诸多细节，严格的测试也是提早发现问题的有效方法。

如果不幸以上环节都被忽略，那么，DBA(也许就是你)就是最后的一环，你必须能够快速的诊断并解决各种复杂问题。

9.3使用 SQL_TRACE/10046 事件进行数据库诊断

SQL_TRACE/10046 事件是 Oracle 提供的用于进行 SQL 跟踪的手段，是强有力的辅助诊断工具.在日常的数据库问题诊断和解决中，SQL_TRACE 是非常常用的方法。当在数据库中启用 SQL_TRACE 或者设置 10046 事件之后，Oracle 将会启动内核跟踪程序，持续记录会话的相关信息，并写入到相应 trace 文件中。跟踪记录的内容包括 SQL 的解析过程、SQL 的执行计划、绑定变量的使用、会话中发生的等待事件等。

在本章之前，我们多次提到和使用过 sql_trace/10046 功能，本节就 SQL_TRACE/10046 事件的使用作简单探讨，并通过具体案例对 sql_trace 的使用进行说明。

9.3.1 SQL_TRACE 及 10046 事件的基础介绍

首先我们先对 SQL_TRACE 及 10046 事件进行一些基本介绍，以使大家能对这个工具有所了解，并熟悉其使用方法。

9.3.1.1 SQL_TRACE 说明

我们先来关注一下 Oracle 官方文档（Oracle9iR2 文档）对 SQL_TRACE 的说明

SQL_TRACE：

参数类型	布尔型
缺省值	false
参数类别	静态

取值范围	true | false

SQL_TRACE 的取值可以启用或禁用 SQL trace 工具。设置 sql_trace 为 true 可以收集信息用于性能优化。DBMS_SYSTEM 包也可以用于实现同样的功能。

警告:

设置初始化参数 SQL_TRACE 为 true 会对整个实例产生严重的性能影响，所以在产品环境中如非必要，确保不要设置这个参数。如果只是对特定的 session 启用跟踪，可以使用 ALTER SESSION 或 DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION 来设置。如果必须在数据库级启用 SQL_TRACE，你需要保证以下条件以最小化性能影响:

1. 至少保证有 25%的 CPU idle
2. 为 USER_DUMP_DEST 分配足够的空间
3. 条带化磁盘以减轻 IO 负担

注意:

如果你使用 ALTER SESSION SET SQL_TRACE 来修改 session 级设置，这个设置并不会在 v$parameter 动态性能视图中体现出来，所以，这个参数仍然被认为是静态参数。

在使用 SQL_TRACE 之前，几个注意事项需要简单说明一下：

1．初始化参数 TIMED_STATISTICS

参数 TIMED_STATISTICS 最好设置为 True，否则一些重要信息不会被收集。

2．设置 MAX_DUMP_FILE_SIZE

该参数设置跟踪文件的大小限制，可以以操作系统块为单位设置；也可以以 K|M 为单位设置；如果跟踪的信息较多，可以干脆设置为 UNLIMITED

从 9i 开始，该参数缺省值为 UNLIMITED。

在 Session 级可以设置如下：

```
SQL> alter session set MAX_DUMP_FILE_SIZE=unlimited;
Session altered.
```

记住前面的警告，你需要足够的空间保存 trace 文件，跟踪过程产生的 Trace 文件可能远远大于你的想象。

SQL_TRACE 可以作为初始化参数在全局启用，也可以通过命令行方式在具体 session 启用。

1．在全局启用 SQL_TRACE

在参数文件(pfile/spfile)中指定:

sql_trace = true

在全局启用 SQL_TRACE 会导致所有进程的活动被跟踪，包括后台进程及所有用户进程，这通常会导致比较严重的性能问题，所以在生产环境中要谨慎使用.

提示：通过在全局启用 sql_trace，我们可以跟踪到所有后台进程的活动，很多在文档中的抽象说明，通过跟踪文件的实时变化，我们可以清晰的看到各个进程之间的紧密协调.

2．在当前 session 级设置

大多数时候我们使用 sql_trace 跟踪当前进程.通过跟踪当前进程可以发现当前操作的后台数据库递归活动(这在研究数据库新特性时尤其有效)，研究 SQL 执行，发现后台错误等.

在 session 级启用和停止 sql_trace 方式如下:

```
启用当前 session 的跟踪:
SQL> alter session set sql_trace=true;
Session altered.
此时的 SQL 操作将被跟踪:
SQL> select count(*) from dba_users;
  COUNT(*)
----------
        34
结束跟踪:
SQL> alter session set sql_trace=false;
Session altered.
```

3．跟踪其他用户进程

在很多时候我们需要跟踪其他用户的进程，而不是当前用户，这可以通过 Oracle 提供的系统包 DBMS_SYSTEM. SET_SQL_TRACE_IN_SESSION 来完成

SET_SQL_TRACE_IN_SESSION 过程序要提供三个参数:

```
SQL> desc dbms_system
…
PROCEDURE SET_SQL_TRACE_IN_SESSION
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --
 SID                            NUMBER                  IN
 SERIAL#                        NUMBER                  IN
 SQL_TRACE                      BOOLEAN                 IN
…
```

通过查询 v$session 我们可以获得 sid、serial#等信息。获得进程信息，选择需要跟踪的进程，设置跟踪：

```
SQL> select sid,serial#,username from v$session where username is not null;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         8       2041 SYS
         9        437 EYGLE
```

设置跟踪:

```
SQL> exec dbms_system.set_sql_trace_in_session(9,437,true)

PL/SQL procedure successfully completed.

….
可以等候片刻，跟踪 session 执行任务,捕获 sql 操作…
如果确定某个功能或模块存在问题，可以在此期间有意识的调用
以确保可以捕获问题代码
….
停止跟踪:
SQL> exec dbms_system.set_sql_trace_in_session(9,437,false)
PL/SQL procedure successfully completed.
```

如果要对其他用户的参数进行设置，我们可能需要用到 DBMS_SYSTEM 包中的另外一个过程：

```
SQL> desc dbms_system
...
PROCEDURE SET_INT_PARAM_IN_SESSION
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SID                            NUMBER                  IN
 SERIAL#                        NUMBER                  IN
 PARNAM                         VARCHAR2                IN
 INTVAL                         BINARY_INTEGER          IN
...
```

比如设置 MAX_DUMP_FILE_SIZE 等参数，可以参考如下：

```
SQL> select sid,serial#,username from v$session where username is not null;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
        18       1605 EYGLE
SQL> begin
  2  sys.dbms_system.set_bool_param_in_session(18, 1605, 'timed_statistics', true);
  3  sys.dbms_system.set_int_param_in_session(18, 1605, 'max_dump_file_size', 2147483647);
  4  sys.dbms_system.set_sql_trace_in_session(18, 1605, true);
  5  end;
  6  /
PL/SQL procedure successfully completed.
```

DBMS_SYSTEM 包功能强大，值得仔细研究。

9.3.1.2 10046 事件说明

10046 事件是 Oracle 提供的内部事件，是对 SQL_TRACE 的增强.

10046 事件可以设置以下四个级别:

1 - 启用标准的 SQL_TRACE 功能,等价于 sql_trace

4 - Level 1 加上绑定值(bind values)

8 - Level 1 + 等待事件跟踪

12 - Level 1 + Level 4 + Level 8

类似 sql_trace，10046 事件可以在全局设置，也可以在 session 级设置。

1．在全局设置

在参数文件中增加:

event="10046 trace name context forever,level 12"

此设置对所有用户的所有进程生效、包括后台进程.

2．对当前 session 设置

通过 alter session 的方式修改，需要 alter session 的系统权限:

```
SQL> alter session set events '10046 trace name context forever, level 8';
Session altered.
SQL> alter session set events '10046 trace name context off';
Session altered.
```

3．对其他用户 session 设置

通过 DBMS_SYSTEM.SET_EV 系统包来实现

```
SQL> desc dbms_system
...
PROCEDURE SET_EV
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 SI                             BINARY_INTEGER          IN
 SE                             BINARY_INTEGER          IN
 EV                             BINARY_INTEGER          IN
 LE                             BINARY_INTEGER          IN
 NM                             VARCHAR2                IN
```

其中的参数 SI、SE 来自 v$session 视图，查询获得需要跟踪的 session 信息:

```
SQL> select sid,serial#,username from v$session where username is not null;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         8       2041 SYS
         9        437 EYGLE
```

执行跟踪:

```
SQL> exec dbms_system.set_ev(9,437,10046,8,'eygle');
PL/SQL procedure successfully completed.
结束跟踪:
SQL> exec dbms_system.set_ev(9,437,10046,0,'eygle');
PL/SQL procedure successfully completed.
```

9.3.1.3　获取跟踪文件

以上生成的跟踪文件位于 user_dump_dest 目录中，位置及文件名可以通过以下 SQL 查询获得:

```
SQL> select
  2     d.value||'/'||lower(rtrim(i.instance, chr(0)))||'_ora_'||p.spid||'.trc' trace_file_name
  3  from
  4     ( select p.spid
  5        from sys.v$mystat m,sys.v$session s,sys.v$process p
  6        where m.statistic# = 1 and s.sid = m.sid and p.addr = s.paddr) p,
  7     ( select t.instance from sys.v$thread    t,sys.v$parameter    v
  8        where v.name = 'thread' and (v.value = 0 or t.thread# = to_number(v.value))) i,
  9     ( select value from sys.v$parameter where name = 'user_dump_dest') d
 10  /

TRACE_FILE_NAME
--------------------------------------------------------------------------------
/opt/oracle/admin/hsjf/udump/hsjf_ora_1026.trc
```

9.3.1.4 读取当前 session 设置的参数

当我们通过 alter session 的方式设置了 sql_trace,这个设置是不能通过 show parameter 的方式得到的,我们需要通过 dbms_system.read_ev 来获取：

```
SQL> set feedback off
SQL> set serveroutput on

SQL> declare
  2     event_level number;
  3  begin
  4     for event_number in 10000..10999 loop
  5        sys.dbms_system.read_ev(event_number, event_level);
  6        if (event_level > 0) then
  7           sys.dbms_output.put_line(
```

```
 8        'Event ' ||
 9        to_char(event_number) ||
10        ' is set at level ' ||
11        to_char(event_level)
12             );
13          end if;
14       end loop;
15    end;
16    /
Event 10046 is set at level 1
```

9.3.2 案例分析之一-隐式转换与索引失效

我们通过几个案例来看一下 SQL_TRACE 在数据库诊断及优化过程中的应用。

9.3.2.1 问题描述

这是帮助一个公司进行优化的诊断案例，应用是一个后台新闻发布系统。前端展现是一个大型网站。JAVA 开发应用，通过中间件连接池连接数据库。

操作系统:SunOS 5.8

数据库版本:8.1.7

系统症状：通过链接访问新闻页极其缓慢，后台发布管理具有同样问题。通常需要十数秒才能返回.

这种性能是用户不能忍受的，需要进行优化，找到问题所在。以下是当时的诊断及问题解决过程，添加了必要的说明，供大家参考。

9.3.2.2 检查并跟踪数据库进程

由于发布系统是非实时系统，诊断时是晚上,基本无用户访问，选择在前台点击相关页面，同时进行后台进程跟踪。

查询 v$session 视图,获取进程信息：

```
SQL> select sid,serial#,username from v$session;
       SID       SERIAL# USERNAME
---------- ---------- ------------------------------
         1          1
         2          1
         3          1
         4          1
         5          1
         6          1
```

```
         7        284 IFLOW
        11        214 IFLOW
        12        164 SYS
        16       1042 IFLOW


10 rows selected.
```

除了 SYS 及后台进程外，其他 3 个进程是我们的诊断目标，对这几个进程启用相关进程 sql_trace:

```
SQL> exec dbms_system.set_sql_trace_in_session(7,284,true)
PL/SQL procedure successfully completed.
SQL> exec dbms_system.set_sql_trace_in_session(11,214,true)
PL/SQL procedure successfully completed.
SQL> exec dbms_system.set_sql_trace_in_session(16,1042,true)
PL/SQL procedure successfully completed.
```

此时在前台对相关页面进行刷新,等候一段时间,关闭 sql_trace

```
SQL> exec dbms_system.set_sql_trace_in_session(7,284,false)
PL/SQL procedure successfully completed.
SQL> exec dbms_system.set_sql_trace_in_session(11,214,false)
PL/SQL procedure successfully completed.
SQL> exec dbms_system.set_sql_trace_in_session(16,1042,false)
PL/SQL procedure successfully completed.
```

9.3.2.3 检查 trace 文件

在 user_dump_dest 目录下，我们可以找到生成的跟踪文件,然后通过 Oracle 提供的格式化工具-tkprof 对 trace 文件进行格式化处理，检查发现以下语句是可疑的：

```
********************************************************************************
select auditstatus,categoryid,auditlevel
from
  categoryarticleassign a,category b where b.id=a.categoryid and articleId=
    20030700400141 and auditstatus>0


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.81       0.81          0       3892          0           1
------- ------  -------- ---------- ---------- ----------  ----------
```

```
total          3       0.81       0.81       0       3892          0          1
```

这里显然是根据 articleId 进行新闻内容读取的，auditstatus>0 应该是限制只能读取审查过的内容。注意这里很可疑的是 query 读取有 3892，按 8k 的 block_size 来计算的话，这大约是 30M 的逻辑读取。这个内容引起了我的注意。

如果遇到过类似的问题，大家在这里就应该可以大致猜到问题的原因了；如果没有遇到过的朋友，可以在这里思考一下再往下看。

这是 Trace 文件里的另外一段:

```
********************************************************************************
select auditstatus,categoryid
from
 categoryarticleassign where articleId=20030700400138 and categoryId in ('63',
  '138','139','140','141','142','143','144','168','213','292','341','346',
  '347','348','349','350','351','352','353','354','355','356','357','358',
  '359','360','361','362','363','364','365','366','367','368','369','370',
  '371','372','383','460','461','462','463','621','622','626','629','631',
  '634','636','643','802','837','838','849','850','851','852','853','854',
  '858','859','860','861','862','863','-1')


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      4.91       4.91          0       2835          7           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      4.91       4.91          0       2835          7           1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 41

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL CATEGORYARTICLEASSIGN
********************************************************************************
```

注意到，这里有一个全表扫描存在，对 categoryarticleassign 表的访问，最终是通过全表扫描完成的。

9.3.2.4 登陆数据库,检查相应索引及表结构

```
SQL> select index_name,table_name,column_name from user_ind_columns
  2  where table_name=upper('categoryarticleassign');

INDEX_NAME                     TABLE_NAME                     COLUMN_NAME
------------------------------ ------------------------------ --------------
IDX_ARTICLEID                  CATEGORYARTICLEASSIGN          ARTICLEID
IND_ARTICLEID_CATEG            CATEGORYARTICLEASSIGN          ARTICLEID
IND_ARTICLEID_CATEG            CATEGORYARTICLEASSIGN          CATEGORYID
IDX_SORTID                     CATEGORYARTICLEASSIGN          SORTID
PK_CATEGORYARTICLEASSIGN       CATEGORYARTICLEASSIGN          ARTICLEID
PK_CATEGORYARTICLEASSIGN       CATEGORYARTICLEASSIGN          CATEGORYID
PK_CATEGORYARTICLEASSIGN       CATEGORYARTICLEASSIGN          ASSIGNTYPE
IDX_CAT_ARTICLE                CATEGORYARTICLEASSIGN          AUDITSTATUS
IDX_CAT_ARTICLE                CATEGORYARTICLEASSIGN          ARTICLEID
IDX_CAT_ARTICLE                CATEGORYARTICLEASSIGN          CATEGORYID
IDX_CAT_ARTICLE                CATEGORYARTICLEASSIGN          ASSIGNTYPE

11 rows selected.
```

我们注意到,该表上 ARTICLEID 字段建有 IDX_ARTICLEID 索引，而该索引在以上查询中都没有被用到.接下来检查表结构:

```
SQL> desc categoryarticleassign
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CATEGORYID                                NOT NULL NUMBER
ARTICLEID                                  NOT NULL VARCHAR2(14)
 ASSIGNTYPE                                NOT NULL VARCHAR2(1)
 AUDITSTATUS                               NOT NULL NUMBER
 SORTID                                    NOT NULL NUMBER
 UNPASS                                             VARCHAR2(255)
```

问题发现:这里 ARTICLEID 是个字符型(VARCHAR2)数据，而在查询中给入的条件是

articleId= 20030700400141

在这个查询中, 20030700400141 被认为是一个数字值。Oracle 在执行这个 SQL 时发生潜在的数据类型转换（把 ARTICLEID 转换为 Number 和 20030700400141 进行比较）,从而导致了索引失效。

我们在 SQL*Plus 中执行类似查询:

```
SQL>   select auditstatus,categoryid
  2    from
```

```
  3       categoryarticleassign where articleId=20030700400132;
AUDITSTATUS CATEGORYID
----------- ----------
          9         94
          0        383
          0        695
Elapsed: 00:00:02.62
Execution Plan
----------------------------------------------------------
   0          SELECT STATEMENT Optimizer=CHOOSE (Cost=110 Card=2 Bytes=38)
   1      0     TABLE ACCESS (FULL) OF 'CATEGORYARTICLEASSIGN' (Cost=110 Card=2 Bytes=38)
```

发现执行的是全表扫描，索引被忽略，这显然不是我们想看到的。

9.3.2.5 解决方法

解决这个问题是简单的,在参数两侧各增加一个单引号（’）,既可解决这个问题。对于用单引号引起来的数字，Oracle 会认为是字符串，这样就消除了隐式类型转换，索引得以被正确使用。

对于类似的查询,我们发现索引被正确使用，Query 模式读取降低为 2，执行该 SQL 几乎不需要花费 CPU 时间了

```
********************************************************************************
select unpass
from
 categoryarticleassign where articleid='20030320000682' and categoryid='113'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          2          0           0
------- ------  -------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0          2          0           0

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 20

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  TABLE ACCESS BY INDEX ROWID CATEGORYARTICLEASSIGN
```

```
      1    INDEX RANGE SCAN (object id 3080)

********************************************************************************
```

至此，这个问题得到了完满的解决。

9.3.2.6 总结

在 Oracle 开发中，我们应该尽量避免使用隐式的数据类型转换，因为隐式数据类型转换可能会带来索引失效的问题，给系统埋下隐患。

这些问题，在开发阶段就应该被避免。使用显示的数据类型转换应该被作为规则确定下来。使用函数导致索引失效的问题与此类似。

我们在很多系统中看到，大量的性能问题都是由于简单的疏忽导致的，而且由于问题的隐蔽性等，这些问题一旦爆发出来，会给诊断优化带来相当的难度，所以完善的规范和良好的编码对于一个系统来说是至关重要的。

9.3.3 案例分析之二-跟踪后台错误

9.3.3.1 问题描述

很多时候，在我们进行数据库操作时，比如 drop user,drop table 等，经常会遇到这样的错误：

```
ORA-00604: error occurred at recursive SQL level 1 .
```

单从这样的提示来看，很多时候是没有丝毫用处的，我们无法确定问题出在何处。本案例就这一类问题提供一个思路及方法供大家参考.

9.3.3.2 drop user 出现问题

这是一个生产环境的数据库，在 Drop User 时报出以下错误后退出

```
ORA-00604: error occurred at recursive SQL level 1
ORA-00942: table or view does not exist .
```

关于 recursive SQL 错误,我们有必要做个简单说明。当我们发出一条简单的 SQL 命令以后,Oracle 数据库要在后台解析这条命令，并转换为 Oracle 数据库的一系列后台操作。这些后台操作统称为递归 SQL。

比如 create table 这样一条简单的 DDL 命令，Oracle 数据库在后台，实际上要把这个命令转换为对于 obj$、tab$、col$等底层表的插入操作；对于 drop table 操作，则是在这些系统表中进行反向删除操作，大家同样可以通过 sql_trace 进行后台跟踪，进一步了解 Oracle 数据库的后台操作。Oracle 所作的工作可能比我们有时候想的要复杂的多。

9.3.3.3 跟踪问题

通过 Oracle 提供 sql_trace 的功能，可以用于跟踪 Oracle 数据库的后台递归操作。研究跟踪文件，可以找到问题的所在，以下是这个问题的跟踪过程:

```
SQL> alter session set sql_trace=true;
Session altered.
SQL> drop user wapcomm;
ORA-00604: error occurred at recursive SQL level 1
ORA-00942: table or view does not exist .
SQL> alter session set sql_trace=false;
```

格式化(使用 tkprof)跟踪文件后，我们获得以下输出(摘录部分):

```
********************************************************************************
The following statement encountered a error during parse:
DELETE FROM SDO_GEOM_METADATA_TABLE    WHERE SDO_OWNER = 'WAPCOMM'
Error encountered: ORA-00942
********************************************************************************

alter session set sql_trace=true

。。。。。。。。。。。。。。。
drop user wapcomm

……

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: SYS
********************************************************************************
.....(省略部分递归 SQL….)

********************************************************************************

delete from user_history$
where
  user# = :1                    -----后台的递归删除操作…

……
Rows     Row Source Operation
```

```
-------    ----------------------------------------------------
         1    DELETE USER_HISTORY$
         1      TABLE ACCESS FULL USER_HISTORY$

********************************************************************************

declare
    stmt varchar2(200);
BEGIN
       if dictionary_obj_type = 'USER' THEN
          stmt := 'DELETE FROM SDO_GEOM_METADATA_TABLE ' ||
                     ' WHERE SDO_OWNER = ''' || dictionary_obj_name || ''' ';
          EXECUTE IMMEDIATE stmt;
      end if;
end;

。。。。。。。

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 31         (recursive depth: 1)
********************************************************************************

alter session set sql_trace=false
……

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: SYS
```

使用 TKPROF 格式化以后，Oracle 把错误信息首先呈现出来。

我们看到 ORA-00942 错误是由于 SDO_GEOM_METADATA_TABLE 表/视图不存在所致，问题由此可以定位。对于这一类的错误，定位问题以后解决的方法就要依据具体问题原因而定了。

9.3.3.4 问题定位

对于本案例，通过 Metalink（http://metalink.oracle.com ）可以确认为一个 Bug，获得以下解释和解决办法：

```
Problem Description
-------------------
```

```
The Oracle Spatial Option has been installed and you are encountering
the following errors while trying to drop a user, who has no spatial tables,
connected as SYSTEM:

ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-00942: table or view does not exist
ORA-06512: at line 7

A 942 error trace shows the failing SQL statement as:

DELETE FROM SDO_GEOM_METADATA_TABLE WHERE SDO_OWNER = '<user>'

Solution Description
--------------------
(1) Create a synonym for SDO_GEOM_METADATA_TABLE under SYSTEM which points to
MDSYS.SDO_GEOM_METADATA_TABLE.
(2) Now the user can be dropped connected as SYSTEM.
```

对于本例，为 MDSYS.SDO_GEOM_METADATA_TABLE 创建一个同义词即可解决.是相对简单的情况。MDSYS.SDO_GEOM_METADATA_TABLE 为 Spatial 对象，如果未使用 Spatial 选项，可以删除

```
SQL> connect / as sysdba
Connected.
SQL> select * from dba_sdo_geom_metadata order by owner;
select * from dba_sdo_geom_metadata order by owner
*
ERROR at line 1:
ORA-00942: table or view does not exist
ORA-04063: view "MDSYS.DBA_SDO_GEOM_METADATA" has errors
SQL> select object_name from dba_objects where object_name like '%SDO%';
OBJECT_NAME
--------------------------------------------------------------------------------
ALL_SDO_GEOM_METADATA
ALL_SDO_INDEX_INFO
ALL_SDO_INDEX_METADATA
DBA_SDO_GEOM_METADATA
DBA_SDO_INDEX_INFO
DBA_SDO_INDEX_METADATA
....
```

```
DBA_SDO_GEOM_METADATA
DBA_SDO_INDEX_INFO
...
88 rows selected.
SQL> drop user MDSYS cascade;
User dropped.
SQL> select owner,type_name from dba_types where type_name like 'SDO%';
no rows selected
SQL> alter session set sql_trace=true;
Session altered.
SQL> drop user wapcomm;
User dropped.
SQL> alter session set sql_trace=false;
Session altered.
```

这时用户得以顺利 drop。

9.3.3.5 一点总结

使用 sql_trace 可以跟踪数据库的很多后台操作，有利于我们发现问题的所在。

很多时候，我们想要研究 Oracle 的内部活动或后台操作,也可以通过 sql_trace 跟踪。这是深入研究学习 Oracle 的必经之途。

9.3.4 10046 与等待事件

如果我们需要获得更多的跟踪信息，就需要用到 10046 事件。如前所述，10046 事件是 SQL_TRACE 功能的增强，可以通过 10046 跟踪获得更多的信息，包括非常有用的等待事件等。

我们知道，在进行数据库问题诊断及性能优化时，经常需要查询的几个重要视图包括：v$session_wait、v$system_event 等，这些视图中主要记录的就是等待事件。

通过调整以降低等待，是提高性能的一个方法。这些等待事件来自所有数据库操作，对于不同进程的等待可以通过动态性能视图 v$session_wait 等来查询；对于数据库全局等待可以通过 v$system_event 等视图来获得。

同样的，我们可以通过对具体 session 的跟踪获得每个 session 的执行情况及等待事件。

9.3.4.1 100046 事件的使用

通过一个简单的测试，我们来看一下 10046 事件的强大之处。

```
SQL> create table t as select * from dba_objects;
Table created.
--创建一张测试表
SQL> select file_id,block_id,blocks from dba_extents where segment_name='T';
```

```
   FILE_ID     BLOCK_ID        BLOCKS
---------- ---------- ----------
         1      21601          8
         1      21609          8
         1      21617          8
         1      21625          8
         1      21633          8
         1      23433          8
         1      23441          8
         1      23449          8
         1      23457          8
         1      23465          8
10 rows selected.
--查看其空间使用情况
SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.
--启用 10046 事件跟踪
SQL> select count(*) from t;
  COUNT(*)
----------
      6207
--由于表上未建立索引,所以此处应该引发一次全表扫描
SQL> alter session set events '10046 trace name context off';
Session altered.
--停用跟踪
```

然后来检查一下 Oracle 生成的跟踪文件，可以获得详细的等待事件及数据读取等信息：

```
[oracle@eygle udump]$ cat rac1_ora_20695.trc |grep scatt
WAIT #1: nam='db file scattered read' ela= 11657 p1=1 p2=21602 p3=7
WAIT #1: nam='db file scattered read' ela= 1363 p1=1 p2=21609 p3=8
WAIT #1: nam='db file scattered read' ela= 1297 p1=1 p2=21617 p3=8
WAIT #1: nam='db file scattered read' ela= 1346 p1=1 p2=21625 p3=8
WAIT #1: nam='db file scattered read' ela= 1313 p1=1 p2=21633 p3=8
WAIT #1: nam='db file scattered read' ela= 6226 p1=1 p2=23433 p3=8
WAIT #1: nam='db file scattered read' ela= 1316 p1=1 p2=23441 p3=8
WAIT #1: nam='db file scattered read' ela= 1355 p1=1 p2=23449 p3=8
WAIT #1: nam='db file scattered read' ela= 1320 p1=1 p2=23457 p3=8
WAIT #1: nam='db file scattered read' ela= 884 p1=1 p2=23465 p3=5
```

大家注意这里的等待事件'db file scattered read',意味着这里使用了全表扫描来访问数据.其中 p1,p2,p3 分别代表了:文件号，起始数据块块号，读取数据块的数量。各参数的含义也可以

从 v$event_name 视图中获得:

```
SQL> select name,PARAMETER1 p1,PARAMETER2 p2,PARAMETER3 p3
  2  from v$event_name  where name='db file scattered read';
NAME                                P1              P2              P3
----------------------------------- ---------- ---------- ----------
db file scattered read              file#           block#          blocks
```

在数据库内部，这些等待时间最后都回累计到 v$system_event 动态性能视图中,是数据库性能诊断的一个重要参考:

```
SQL> select event,time_waited from v$system_event where event='db file scattered read';
EVENT                                                            TIME_WAITED
---------------------------------------------------------------- -----------
db file scattered read                                                    51
```

9.3.4.2 10046 与 db_file_multiblock_read_count

这里有必要提到另外一个相关的初始化参数: **db_file_multiblock_read_count,**这个参数代表 Oracle 在执行全表扫描时每次 IO 操作可以读取得数据块的数量.

在前面的测试中，我的 db_file_multiblock_read_count 参数设置为 16，由于 extent 大小为 8 个 block，Oracle 的一次 IO 操作不能跨越 extent，所以前面的全表扫描每次只能读取 8 个 block，进行了 10 次 IO 读取。

来看一下进一步的测试，首先创建一个本地管理的表空间并创建测试表：

```
SQL> create tablespace eygle
  2  datafile '/dev/raw/raw2' size 100M
  3  extent management local uniform size 256K;
Tablespace created.
SQL> alter table t move tablespace eygle;
Table altered.
SQL> select file_id,block_id,blocks from dba_extents where segment_name='T';
   FILE_ID   BLOCK_ID     BLOCKS
---------- ---------- ----------
         4          9         32
         4         41         32
         4         73         32
```

执行全表访问并跟踪后台操作：

```
SQL> show parameter read_count
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_multiblock_read_count        integer     16
SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.
```

```
SQL> select count(*) from t;
  COUNT(*)
----------
      6207
SQL> alter session set events '10046 trace name context off';
Session altered.
SQL>@gettracname
TRACE_FILE_NAME
--------------------------------------------------------------------------------
/opt/oracle/admin/rac/udump/rac1_ora_20912.trc
SQL> !
[oracle@eygle rac]$ cat /opt/oracle/admin/rac/udump/rac1_ora_20912.trc |grep scatt
WAIT #1: nam='db file scattered read' ela= 12170 p1=4 p2=10 p3=16
WAIT #1: nam='db file scattered read' ela= 2316 p1=4 p2=26 p3=15
WAIT #1: nam='db file scattered read' ela= 2454 p1=4 p2=41 p3=16
WAIT #1: nam='db file scattered read' ela= 2449 p1=4 p2=57 p3=16
WAIT #1: nam='db file scattered read' ela= 2027 p1=4 p2=73 p3=13
```

观察输出，此时 Oracle 只需要 5 次 IO 操作就完成了全表扫描。通常较大的 db_file_multiblock_read_count 设置可以加快全表扫描的执行，但是根据经验大于 32 的设置通常不会带来更大的性能提高。

9.3.4.3 10046 与执行计划的选择

而且需要注意的是，增大 db_file_multiblock_read_count 参数的设置，会使全表扫描的成本降低，在 CBO 优化器下可能会使 Oracle 更倾向于使用全表扫描而不是索引访问.

看一下进一步测试:

```
SQL> select owner,count(*) from t group by owner;

OWNER                          COUNT(*)
------------------------------ ----------
OUTLN                               7
PUBLIC                           1623
SYS                              4042
SYSTEM                            404
WMSYS                             131
SQL> create index i_owner on t(owner);
Index created.
SQL> analyze table t compute statistics for table
  2          for all indexes
  3          for all indexed columns;
```

```
Table analyzed.
SQL> set autotrace traceonly explain
SQL> alter session set db_file_multiblock_read_count=16;
Session altered.
SQL> select * from t where owner='SYSTEM';
Execution Plan
----------------------------------------------------------
   0        SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=404 Bytes=34744)
   1    0     TABLE ACCESS (BY INDEX ROWID) OF 'T' (Cost=7 Card=404 Bytes=34744)
   2    1       INDEX (RANGE SCAN) OF 'I_OWNER' (NON-UNIQUE) (Cost=1 Card=404)
SQL> alter session set db_file_multiblock_read_count=32;
Session altered.
SQL> select * from t where owner='SYSTEM';
Execution Plan
----------------------------------------------------------
   0        SELECT STATEMENT Optimizer=CHOOSE (Cost=6 Card=404 Bytes=34744)
   1    0     TABLE ACCESS (FULL) OF 'T' (Cost=6 Card=404 Bytes=34744)
```

我们注意到，当增大 **db_file_multiblock_read_count** 参数时，全表扫描的成本降低，Oracle 在后面的执行计划中选择了全表扫描。所以，当你修改这个初始化参数时，你必须认识到，很多 SQL 的执行计划可能就此改变。

9.3.4.4 db_file_multiblock_read_count 与系统的 IO 能力

db_file_multiblock_read_count 的设置要受 OS 最大 IO 能力影响，也就是说，如果你系统的硬件 IO 能力有限，即使设置再大的 db_file_multiblock_read_count 也是没有用的。理论上，最大 db_file_multiblock_read_count 和系统 IO 能力应该有如下关系:

```
Max(db_file_multiblock_read_count) = MaxOsIOsize/db_block_size
```

当然这个 Max(db_file_multiblock_read_count)还受 Oracle 的限制。SSTIOMAX 是 Oracle 的内部参数或常数，用以限制单次 IO 读写操作的最大数据传输量。

这个参数是固定的，不能被修改。

所以 db_file_multiblock_read_count 参数的设置就会受到 SSTIOMAX 的限制:

```
db_block_size * db_file_multiblock_read_count <= SSTIOMAX
```

另外一个限制是最大 db_file_multiblock_read_count 值不能超过 db_block_buffers/4。

可以通过 db_file_multiblock_read_count 来测试 Oracle 在不同系统下，单次 IO 最大所能读取得数据量：

```
SYS AS SYSDBA on 11-AUG-04 >show parameter read_count
NAME                                 TYPE             VALUE
------------------------------------ ---------- ------------------------------
db_file_multiblock_read_count        integer        16
SYS AS SYSDBA on 11-AUG-04 >create tablespace dfmbrc
```

```
  2    datafile '/opt/oracle/oradata/eygle/dfmbrc.dbf'
  3    size 20M extent management local uniform size 2M;
Tablespace created.
SYS AS SYSDBA on 11-AUG-04 >create table t tablespace dfmbrc as select * from dba_objects;
Table created.
SYS AS SYSDBA on 11-AUG-04 >insert into t select * from t;
9149 rows created.
SYS AS SYSDBA on 11-AUG-04 >/
36596 rows created.
SYS AS SYSDBA on 11-AUG-04 >commit;
Commit complete.
SYS AS SYSDBA on 11-AUG-04 >alter session set db_file_multiblock_read_count=1000;
Session altered.
SYS AS SYSDBA on 12-AUG-04 >show parameter read_count
NAME                                 TYPE             VALUE
------------------------------------ ---------- ------------------------------
db_file_multiblock_read_count        integer          128
SYS AS SYSDBA on 11-AUG-04 >alter session set events '10046 trace name context forever,level 12';
Session altered.
SYS AS SYSDBA on 11-AUG-04 >alter system flush buffer_cache;
System altered.
SYS AS SYSDBA on 11-AUG-04 >select count(*) from t;
  COUNT(*)
----------
     73192
SYS AS SYSDBA on 12-AUG-04 >@gettrace
TRACE_FILE_NAME
--------------------------------------------------------------------------------
/opt/oracle/soft/eygle_ora_24432.trc
$ cat /opt/oracle/soft/eygle_ora_24432.trc|grep sca
WAIT #26: nam='db file scattered read' ela= 18267 p1=10 p2=10 p3=128
WAIT #26: nam='db file scattered read' ela= 8836 p1=10 p2=138 p3=127
WAIT #26: nam='db file scattered read' ela= 8923 p1=10 p2=265 p3=128
WAIT #26: nam='db file scattered read' ela= 8853 p1=10 p2=393 p3=128
WAIT #26: nam='db file scattered read' ela= 8985 p1=10 p2=521 p3=128
WAIT #26: nam='db file scattered read' ela= 8997 p1=10 p2=649 p3=128
WAIT #26: nam='db file scattered read' ela= 9096 p1=10 p2=777 p3=128
WAIT #26: nam='db file scattered read' ela= 583 p1=10 p2=905 p3=128
```

我们可以看到，在以上测试平台中，Oracle 最多每次 IO 能够读取 128 个 Block，由于

block_size 为 8k,也就是每次最多读取了 1M 数据。系统平台为:

```
$ uname -a
SunOS billing 5.8 Generic_108528-23 sun4u sparc SUNW,Ultra-4
```

9.3.4.5 总结

Sql_trace/10046 事件是 Oracle 提供的非常强大的工具及手段，我们应该深入了解、掌握并且熟练运用它.希望本文能够带大家了解这个事件，而怎样使用它去解决问题，了解 Oracle，还有待大家进一步的探索.

9.4使用物化视图进行翻页性能调整

物化视图从 Oracle8i 被引入到数据库中，最初被作为数据仓库/决策支持系统的工具，是概要管理的一部分。物化试图通过预计算或汇总构建自己的独立存储，从而可以极大的提高相关处理的性能，通过查询重写（Query Rewrite）功能，Oracle 可以自动对 SQL 进行改写以最大程度的发挥物化视图的作用。

物化视图是典型的通过存储空间换取性能的方式，通过物化视图，Oracle 可以：

1．有效的减少逻辑读取

2．减少写操作--通过消除排序及聚集实现

3．减少 CPU 的消耗--无需实时进行复杂运算

4．显著提高响应速度

这是物化视图的主要特点。接下来通过一个具体案例的应用，说明物化视图在优化中的应用。

9.4.1 系统环境

OS：Linux AD2.1

数据库版本：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production
```

9.4.2 问题描述

数据库系统出现如下错误：

```
Mon Dec  6 16:51:44 2004
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
Mon Dec  6 16:51:55 2004
```

```
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
Mon Dec    6 16:52:51 2004
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
Mon Dec    6 16:52:52 2004
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
Mon Dec    6 16:52:54 2004
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
Mon Dec    6 16:52:55 2004
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
Mon Dec    6 16:52:56 2004
ORA-1652: unable to extend temp segment by 128 in tablespace                TEMP2
```

临时表空间不能扩展，前台应用出现异常，研发人员请求协助，开始介入进行诊断。

9.4.3 捕获排序 SQL 语句

首先尝试捕获引发排序的 SQL 语句，使用以下代码 getsort.sql

```
SELECT              /*+ rule */
       DISTINCT a.SID, a.process, a.serial#,
                TO_CHAR (a.logon_time, 'YYYYMMDD HH24:MI:SS') LOGON, a.osuser,
                TABLESPACE, b.sql_text
           FROM v$session a, v$sql b, v$sort_usage c
          WHERE a.sql_address = b.address(+) AND a.sql_address = c.sqladdr
/
```

获得以下主要排序语句：

```
SQL> @getsort
       SID    PROCESS                      SERIAL#    LOGON                                OSUSER
TABLESPACE
---------- ------------ ---------- ----------------- ------------------------------------
SQL_TEXT
--------------------------------------------------------------------------------
        15                          24965 20041207 16:38:01 oracle                           TEMP2
select count(1) from HW_User4Love u where u.numIntention<>99 and u.numGender=:1
        21                          49757 20041207 16:33:28 oracle                           TEMP2
select * from (select t.*, rownum i from (select
u.numUserId,u.vc2UserName,u.numUserType,u.numRank,u.numGender,u.numAge,
u.numDistrict,u.vc2District,u.numLooking,u.numPersonality,u.numAbility,u.numZodiac,u.numExperience
from HW_User4Love u where u.numIntention<>99 order by u.numUserType desc, u.numRank desc,
u.numUserId desc) t where rownum<=260) where i>240
        30                          65414 20041207 16:34:04 oracle                           TEMP2
```

```
select * from (select t.*, rownum i from (select u.numUserId,u.vc2UserName,u.numUserType,u.numRank,u.numGender,u.numAge,
u.numDistrict,u.vc2District,u.numLooking,u.numPersonality,u.numAbility,u.numZodiac,u.numExperience
from HW_User4Love u where u.numIntention<>99 order by u.numUserType desc, u.numRank desc,
u.numUserId desc) t where rownum<=40) where i>20
```

大量类似的 SQL 在占用大量的排序空间，确定是这些 SQL 引起的临时表空间过量使用。这些 SQL 需要研究和优化。

9.4.4 确定典型问题 SQL

主要问题 SQL，显然是一个翻页查询程序：

```
SELECT *
  FROM (SELECT t.*, ROWNUM i
          FROM (SELECT     u.numuserid, u.vc2username, u.numusertype, u.numrank,
                           u.numgender, u.numage, u.numdistrict, u.vc2district,
                           u.numlooking, u.numpersonality, u.numability,
                           u.numzodiac, u.numexperience
                      FROM hw_user4love u
                     WHERE u.numintention <> 99 AND u.numgender = :1
                  ORDER BY u.numusertype DESC, u.numrank DESC, u.numuserid DESC) t
         WHERE ROWNUM <= 40)
 WHERE i > 20
```

查看该 SQL 语句的执行计划：

```
SQL> @hawa
20 rows selected.
Elapsed: 00:01:23.92
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=31692 Card=40 Bytes=7680)
   1    0   VIEW (Cost=31692 Card=40 Bytes=7680)
   2    1     COUNT (STOPKEY)
   3    2       VIEW (Cost=31692 Card=582622 Bytes=104289338)
   4    3         SORT (ORDER BY STOPKEY) (Cost=31692 Card=582622 Bytes=30296344)
   5    4           HASH JOIN (Cost=7435 Card=582622 Bytes=30296344)
   6    5             HASH JOIN (Cost=4991 Card=583296 Bytes=20415360)
   7    6               TABLE ACCESS (FULL) OF 'HW_USERPROFILE' (Cost=1752 Card=583296
Bytes=11082624)
   8    6               TABLE ACCESS (FULL) OF 'HW_USER' (Cost=1799 Card=1380038
Bytes=22080608)
```

```
   9      5                    TABLE ACCESS (FULL) OF 'HW_USERSCORE' (Cost=506 Card=1332871
Bytes=22658807)

Statistics
----------------------------------------------------------
          0  recursive calls
        255  db block gets
      44760  consistent gets
      70299  physical reads
          0  redo size
       2246  bytes sent via SQL*Net to client
        514  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
         20  rows processed
```

发现该SQL调用了三个底层表，执行全表扫描，逻辑读高达：**44760** 。这里HW_User4Love实际上是一个视图。查询其创建语句：

```
SQL> select text from dba_views where view_name=upper('HW_User4Love');
TEXT
--------------------------------------------------------------------------------
select u.numUserId as numUserId, u.vc2UserName as vc2UserName, u.numUserType as numUserType,
   u.vc2MobileNumber as vc2MobileNumber,u.numMobileStatus as numMobileStatus,
   u.vc2Mail as vc2Mail, u.numMailStatus as numMailStatus,
   s.numRank as numRank, s.numExperience as numExperience,
   p.numGender as numGender, p.datBirthday as datBirthday, p.numAge, p.numZodiac,
   p.numDistrict as numDistrict,p.vc2District as vc2District, p.numHeight as numHeight,
   p.numSmoke as numSmoke, p.numDrink as numDrink, p.chrInterests as chrInterests,
   p.numStatus as numIntention,s.numLooking as numLooking, s.numPersonality as numPersonality,
   s.numAbility as numAbility
from    HW_User u,    HW_UserProfile p,    HW_UserScore s
where    u.numUserId = p.numUserId and u.numUserId = s.numUserId and s.numExperience > 100
with read only
Elapsed: 00:00:00.64
```

而三个底层表都有大量记录：

```
SQL> select count(*) from hw_user;
  COUNT(*)
----------
   1378484
```

```
SQL> select count(*) from hw_userprofile;
  COUNT(*)
----------
   1378470
SQL> select count(*) from hw_userscore;
  COUNT(*)
----------
   1378498
```

这三个表的全表扫描对数据库的性能产生了巨大的冲击。进一步确认发现，在这个结果集中符合视图查询条件的记录只有少量:

```
SQL> select count(*) from hw_user4love;
  COUNT(*)
----------
    234975
```

显然每次通过视图全表扫描三个底层表是产生性能问题的主要原因。

9.4.5　选择解决办法

结合业务逻辑，考虑创建物化视图，通过物化视图的中间存储消除不必要的全表扫描。创建物化视图如下：

```
CREATE MATERIALIZED VIEW HW_User4Love
       BUILD IMMEDIATE
       REFRESH COMPLETE START WITH SYSDATE
       NEXT    trunc(sysdate+1) +4/24
       ENABLE QUERY REWRITE
       AS
select
  u.numUserId as numUserId,
  u.vc2UserName as vc2UserName,
  u.numUserType as numUserType,
  u.vc2MobileNumber as vc2MobileNumber,
  u.numMobileStatus as numMobileStatus,
  u.vc2Mail as vc2Mail,
  u.numMailStatus as numMailStatus,
  s.numRank as numRank,
  s.numExperience as numExperience,
  p.numGender as numGender,
  p.datBirthday as datBirthday,
  p.numAge,
```

```
  p.numZodiac,
  p.numDistrict as numDistrict,
  p.vc2District as vc2District,
  p.numHeight as numHeight,
  p.numSmoke as numSmoke,
  p.numDrink as numDrink,
  p.chrInterests as chrInterests,
  p.numStatus as numIntention,
  s.numLooking as numLooking,
  s.numPersonality as numPersonality,
  s.numAbility as numAbility
from
  HW_User u,
  HW_UserProfile p,
  HW_UserScore s
where
  u.numUserId = p.numUserId and u.numUserId = s.numUserId and s.numExperience > 100
```

注意，你必须充分考虑你的业务需求，是否允许足够的刷新间隔，我定义的是每日刷新一次，在凌晨 4 点进行一次完全刷新。然后我们再来看类似查询：

```
SQL> @hawa
20 rows selected.
Elapsed: 00:00:01.03
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2381 Card=40 Bytes=7680)
   1    0   VIEW (Cost=2381 Card=40 Bytes=7680)
   2    1     COUNT (STOPKEY)
   3    2       VIEW (Cost=2381 Card=102802 Bytes=18401558)
   4    3         SORT (ORDER BY STOPKEY) (Cost=2381 Card=102802 Bytes=4523288)
   5    4           TABLE ACCESS (FULL) OF 'HW_USER4LOVE' (Cost=337 Card=102802
Bytes=4523288)

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       3446  consistent gets
       2048  physical reads
          0  redo size
```

```
      2246  bytes sent via SQL*Net to client
       514  bytes received via SQL*Net from client
         3  SQL*Net roundtrips to/from client
         1  sorts (memory)
         0  sorts (disk)
        20  rows processed
```

我们注意到，现在这个查询在 1 秒左右即可执行完毕，逻辑读从原来的：**44760** 减少到现在的 **3446**，性能大大提高。

> 提示：有效的降低逻辑读是 SQL 优化的基本原则之一。

9.4.6　进一步的调整优化

注意到这里仍然对 HW_USER4LOVE 物化视图执行了全表扫描，我们可以通过对其创建索引来进行进一步的优化。

在原 SQL 语句中包含：

order by u.numUserType desc, u.numRank desc, u.numUserId desc

正是这个 order by 子句导致了排序，可以通过创建降序索引消除这个排序：

```
SQL> create index idx_desc on HW_USER4LOVE (numUserType desc, numRank desc, numUserId desc);
Index created.

Elapsed: 00:00:04.48
```

> 注意：降序索引本质上是基于函数的索引，只有在 CBO 下才能被用到。
>
> Connected to Oracle9i Enterprise Edition Release 9.2.0.4.0
>
> Connected as hawa
>
> SQL> create table t as select * from dba_users;
>
> Table created
>
> SQL> create index idx_username_desc on t(username desc);
>
> Index created
>
> SQL> select index_name,table_name,INDEX_TYPE from user_indexes where table_name='T';
>
> INDEX_NAME　　TABLE_NAME
> INDEX_TYPE

```
------------------------------ ------------------------------ --------------------------

IDX_USERNAME_DESC                                                                T
        FUNCTION-BASED NORMAL

SQL> select column_name,column_position,descend from user_ind_columns

      2       where table_name='T';

COLUMN_NAME                    COLUMN_POSITION DESCEND

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

SYS_NC00013$                                   1                        DESC

注 意 ， 降 序 索 引 以 及   FBI   的 定 义 可 以 从   DBA_IND_EXPRESSIONS   或
        USER_IND_EXPRESSIONS 视图中获得。

SQL> select * from user_ind_expressions where table_name='T';



INDEX_NAME                                TABLE_NAME             COLUMN_EXPRESSION
        COLUMN_POSITION

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

IDX_USERNAME_DESC              T                                                "USERNAME"
        1
```

再次执行原查询语句：

```
SQL> @hawa
20 rows selected.
Elapsed: 00:00:00.22

Execution Plan
----------------------------------------------------------
   0          SELECT STATEMENT Optimizer=CHOOSE (Cost=826 Card=40 Bytes=7680)
   1      0     VIEW (Cost=826 Card=40 Bytes=7680)
   2      1       COUNT (STOPKEY)
   3      2         VIEW (Cost=826 Card=102802 Bytes=18401558)
   4       3           TABLE ACCESS (BY INDEX ROWID) OF 'HW_USER4LOVE' (Cost=826
Card=102802 Bytes=4523288)
   5      4             INDEX (FULL SCAN) OF 'IDX_DESC' (NON-UNIQUE) (Cost=26 Card=234975)
```

```
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         88  consistent gets
          2  physical reads
          0  redo size
       2246  bytes sent via SQL*Net to client
        514  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
         20  rows processed
```

现在 Oracle 通过 Index Full Scan 来执行以上查询，逻辑读下降到了：**88**，而且排序被彻底消除了。问题至此算是有了一个较好的解决。

9.4.7　一点总结

Oracle 的物化试图功能强大，使用起来也有很多需要注意的地方，最重要的是，你的业务逻辑是否允许。所以怎样使用它，还取决于你对 Oracle 以及自己业务的了解。

关于物化视图的查询重写功能，Thomas Kyte 在他的书中有详细的描述，在我的站点上（www.eygle.com ）上也有详细描述，文中不再赘述。

9.5 一次横跨两岸的问题诊断

最近，通过邮件的方式协助一位朋友解决了一个性能问题，其过程和判断很有参考价值，简单收录在这里，供大家参考，值得高兴的是，网络是没有边界的。

9.5.1　第一封求助邮件

第一封收到的邮件，这位朋友这样描述：

拜讀您在 網路上所寫 Oracle 诊断案例-SGA 与 Swap 之二，相信您一定可以協助在下所碰到的問題。我負責的一個系統出現了很高的 iowait ，僅知應該是 oracle 造成。

只要 oracle 開啟，就會造成 iowait 升高，而且執行速度非常緩慢，另外，這是今天才發生的，之前也曾發生一次，但是把 sga 調小後就恢復正常。但是隔了一個多月，現在發生再怎麼調整卻還是無法恢復原有的效能。以下相關資訊如下：

top 結果如下

```
last pid: 20888;    load averages:    2.13,    2.50,    2.50
349 processes: 346 sleeping, 1 zombie, 2 on cpu
CPU states: 53.7% idle, 18.6% user, 10.7% kernel, 16.9% iowait,    0.0% swap
Memory: 64G real, 46G free, 10G swap in use, 51G swap free

   PID USERNAME THR PRI NICE    SIZE      RES STATE       TIME       CPU COMMAND
  3589 snims      9    59       0      47M      36M sleep       0:23    0.69% PmTom
  2569 oracle     1    59       0 8599M 8557M sleep       1:04    0.61% oracle
  3444 snims      1    50       0 1520K 1248K sleep       3:08    0.57% logControl
  2571 oracle     1    56       0 8599M 8557M sleep       1:04    0.55% oracle
  2573 oracle     1    59       0 8599M 8557M sleep       1:00    0.55% oracle
  2567 oracle     1    56       0 8599M 8557M sleep       1:09    0.52% oracle
 10907 snims      6    59       0      27M      16M sleep       0:12    0.47% PmGW
  3599 snims      1    59       0 8599M 8560M sleep       0:59    0.45% oracle
 26045 snims      1    59       0 2920K 1840K cpu/18      0:23    0.32% top
 13836 snims      1    59       0 2952K 1856K sleep       0:46    0.31% top
  1094 oracle    15    59       0 8605M 8554M sleep       1:49    0.27% oracle
  3443 snims      7    59       0      33M      21M sleep       1:17    0.25% AlarmGW
  3361 snims      1    59       0 8599M 8559M sleep       9:18    0.17% oracle
  7440 snims      1    59       0 2352K 1864K sleep       0:05    0.16% iPT
```

透過 sar 可以看到 oracle 所在的硬盤 busy 是 99%

```
->sar -d 2 3
SunOS tps2snims11 5.8 Generic_117350-05 sun4u       04/10/06
15:56:02      device        %busy     avque     r+w/s    blks/s    avwait    avserv
15:56:04     md0               10       0.1        20       327       0.0       5.0
               ssd13           99       1.9       108      1501       0.0      18.0
             ssd13,a            0       0.0         0         0       0.0       0.0
             ssd13,b            0       0.0         0         0       0.0       0.0
             ssd13,c            0       0.0         0         0       0.0       0.0
             ssd13,d           99       1.9       108      1501       0.0      18.0
             ssd13,e            0       0.0         0         0       0.0       0.0
```

而 sga 顯示結果如下：

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.4.0 - 64bit Production
JServer Release 9.2.0.4.0 - Production


SQL> show sga;
```

```
Total System Global Area 8936995304 bytes
Fixed Size                  743912 bytes
Variable Size              2097152000 bytes
Database Buffers           6828326912 bytes
Redo Buffers                10772480 bytes
```

alert.log 會一直出現如下的錯誤訊息

```
Creating archive destination LOG_ARCHIVE_DEST_1: '/opt/oracle/oradata/arc/430714142_1_7965.arc'
Mon Apr 10 09:36:46 2006
ARC0: Completed archiving    log 4 thread 1 sequence 7965
Mon Apr 10 09:40:29 2006
Thread 1 cannot allocate new log, sequence 7967
Checkpoint not complete
   Current log# 5 seq# 7966 mem# 0: /opt/oracle/oradata/SNIMS/redo05a.log
   Current log# 5 seq# 7966 mem# 1: /opt/oradata/redo05b.log
Mon Apr 10 09:44:30 2006
Thread 1 advanced to log sequence 7967
   Current log# 2 seq# 7967 mem# 0: /opt/oracle/oradata/SNIMS/redo02a.log
   Current log# 2 seq# 7967 mem# 1: /opt/oradata/redo02b.log
Mon Apr 10 09:44:30 2006
ARC1: Evaluating archive      log 5 thread 1 sequence 7966
ARC1: Beginning to archive log 5 thread 1 sequence 7966
Creating archive destination LOG_ARCHIVE_DEST_2: 'SNIMS_TPS1SNIMS11'
Creating archive destination LOG_ARCHIVE_DEST_1: '/opt/oracle/oradata/arc/430714142_1_7966.arc'
Mon Apr 10 09:44:51 2006
ARC1: Completed archiving    log 5 thread 1 sequence 7966
Mon Apr 10 09:45:46 2006
Thread 1 cannot allocate new log, sequence 7968
Checkpoint not complete
   Current log# 2 seq# 7967 mem# 0: /opt/oracle/oradata/SNIMS/redo02a.log
   Current log# 2 seq# 7967 mem# 1: /opt/oradata/redo02b.log
Mon Apr 10 09:50:26 2006
Thread 1 advanced to log sequence 7968
   Current log# 3 seq# 7968 mem# 0: /opt/oracle/oradata/SNIMS/redo03a.log
   Current log# 3 seq# 7968 mem# 1: /opt/oradata/redo03b.log
Mon Apr 10 09:50:26 2006
ARC0: Evaluating archive      log 2 thread 1 sequence 7967
ARC0: Beginning to archive log 2 thread 1 sequence 7967
Creating archive destination LOG_ARCHIVE_DEST_2: 'SNIMS_TPS1SNIMS11'
Creating archive destination LOG_ARCHIVE_DEST_1: '/opt/oracle/oradata/arc/430714142_1_7967.arc'
```

```
Mon Apr 10 09:50:45 2006
ARC0: Completed archiving    log 2 thread 1 sequence 7967
Mon Apr 10 09:51:38 2006
Thread 1 cannot allocate new log, sequence 7969
Checkpoint not complete
  Current log# 3 seq# 7968 mem# 0: /opt/oracle/oradata/SNIMS/redo03a.log
  Current log# 3 seq# 7968 mem# 1: /opt/oradata/redo03b.log
Mon Apr 10 09:59:13 2006
Thread 1 advanced to log sequence 7969
  Current log# 1 seq# 7969 mem# 0: /opt/oracle/oradata/SNIMS/redo01a.log
  Current log# 1 seq# 7969 mem# 1: /opt/oradata/redo01b.log
Mon Apr 10 09:59:13 2006
ARC1: Evaluating archive      log 3 thread 1 sequence 7968
ARC1: Beginning to archive log 3 thread 1 sequence 7968
Creating archive destination LOG_ARCHIVE_DEST_2: 'SNIMS_TPS1SNIMS11'
Creating archive destination LOG_ARCHIVE_DEST_1: '/opt/oracle/oradata/arc/430714142_1_7968.arc'
Mon Apr 10 09:59:32 2006
ARC1: Completed archiving    log 3 thread 1 sequence 7968
Mon Apr 10 10:00:28 2006
Thread 1 cannot allocate new log, sequence 7970
Checkpoint not complete
  Current log# 1 seq# 7969 mem# 0: /opt/oracle/oradata/SNIMS/redo01a.log
  Current log# 1 seq# 7969 mem# 1: /opt/oradata/redo01b.log
```

而我們的 log 資訊如下

```
SQL> select * from v$log;
GROUP#  THREAD#    SEQUENCE#          BYTES  MEMBERS  ARC  STATUS     FIRST_CHANGE#
FIRST_TIM
------ ------- ---------- ---------- ------- --- -------- ------------- ---------
     1       1       8004   52428800        2 YES INACTIVE       541861070 10-APR-06
     2       1       8002   52428800        2 YES INACTIVE       541764248 10-APR-06
     3       1       8003   52428800        2 YES INACTIVE       541766356 10-APR-06
     4       1       8005  104857600        2 YES INACTIVE       541897189 10-APR-06
     5       1       8006  104857600        2 NO  CURRENT        541923086 10-APR-06
```

而 /etc/system 相關資訊如下：

```
forceload: misc/obpsym
set nopanicdebug = 1
set TS:ts_sleep_promote=1
set shmsys:shminfo_shmmax=10737418240
set shmsys:shminfo_shmmin=100
```

```
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmseg=100
set semsys:seminfo_semmni=4096
set semsys:seminfo_semmsl=256
set semsys:seminfo_semmns=4096
set semsys:seminfo_semopm=100
set semsys:seminfo_semvmx=32767
```

綜合以上狀態，不知道您是否能夠瞭解我們的問題？

从这封邮件的信息，我们可以做出如下一些判断：

1．Checkpoint not complete 提示

说明系统的 I/O 写出存在问题，这可能是由于数据库过于繁忙，生成日志过多，也有可能是因为存储的 I/O 能力存在问题

2．Sar 的输出

数据文件存放路径 99%的 I/O Wait，进一步说明 I/O 负荷沉重，系统经历 I/O 等待。

3．V$log 输出来自不同时段

V$log 输出中，多数日志组处于 INACTIVE 状态，所以和 alert 文件显然来自不同时段。

9.5.2 第一次回复

我这样回复他：

从 v$log 情况来看,你的 Checkpoint not complete 问题应该是间发的,并非频繁.

能否提供一段连续的 iostat 采样信息,让我看一下 io 状况。

另外,根据你的状况,极有可能是应用存在问题。在问题出现时,请查询 v$session_wait 信息给我参考.

9.5.3 进一步信息提供

网友回信，提供了 iostat 信息：

```
->iostat -xtcz 2 10
                    extended device statistics                  tty          cpu
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b  tin tout  us sy wt id
md0          0.0   45.4    0.0  363.3  0.0  0.3    6.1   0  28    3 1023   5  5 17 72
md10         0.0   45.4    0.0  363.3  0.0  0.3    5.5   0  25
md20         0.0   45.9    0.0  367.3  0.0  0.2    3.6   0  17
sd0          0.0   45.4    0.0  363.3  0.0  0.2    5.5   0  25
sd1          0.0   45.9    0.0  367.3  0.0  0.2    3.6   0  17
ssd13        0.0   83.3    0.0  669.9  0.0  1.2   14.1   0  99
                    extended device statistics                  tty          cpu
```

```
device          r/s     w/s     kr/s     kw/s wait actv   svc_t  %w  %b  tin tout   us sy wt id
md0             0.0    51.0      0.0    408.0  0.0  0.3    6.0   0  31    3  692    3  9 16 71
md10            0.0    51.0      0.0    408.0  0.0  0.2    4.7   0  24
md20            0.0    50.5      0.0    404.0  0.0  0.2    4.1   0  21
sd0             0.0    51.0      0.0    408.0  0.0  0.2    4.7   0  24
sd1             0.0    50.5      0.0    404.0  0.0  0.2    4.1   0  20
ssd13           0.0    81.0      0.0    647.5  0.0  1.1   13.1   0  99
                        extended device statistics                      tty          cpu
device          r/s     w/s     kr/s     kw/s wait actv   svc_t  %w  %b  tin tout   us sy wt id
md0             0.0     1.0      0.0      8.0  0.0  0.0   13.5   0   1    3  735    4  2 13 82
md10            0.0     1.0      0.0      8.0  0.0  0.0   13.3   0   1
md20            0.0     1.0      0.0      8.0  0.0  0.0   10.7   0   1
sd0             0.0     1.5      0.0      8.2  0.0  0.0   12.4   0   2
sd1             0.0     1.5      0.0      8.2  0.0  0.0   10.6   0   2
ssd13           0.0    90.0      0.0    767.2  0.0  1.1   12.7   0  99
```

在这个输出中，注意到在磁盘 I/O 繁忙度为 99%时，写出能力却仅有 767 k/s 左右，这个 I/O 能力是很低的。I/O 能力低下是造成系统性能问题的主要原因。而 I/O 能力低下又可能有几方面的原因，第一可能是硬件本身的限制；第二可能是硬件异常导致 I/O 能力受限。

还有 v$session_wait 查询输出，摘录部分内容:

```
SQL> select * from v$session_wait;

       SID        SEQ# EVENT
---------- ---------- ----------------------------------------------------
        31       4634 enqueue
        44       4578 enqueue
       206        206 enqueue
       204       5631 enqueue
       195        195 enqueue
       141        209 enqueue
       129        180 enqueue
       125        175 enqueue
        59       4571 enqueue
        25        460 buffer busy waits
       151        107 buffer busy waits
       191       1217 buffer busy waits
       140       3321 buffer busy waits
       127        439 buffer busy waits
       126        763 buffer busy waits
        82        502 buffer busy waits
```

```
        39        274 log file switch (checkpoint incomplete)
       100       4972 log file switch (checkpoint incomplete)
       197         64 log file switch (checkpoint incomplete)
       187      41946 log file switch (checkpoint incomplete)
       156        337 log file switch (checkpoint incomplete)
       107        317 log file switch (checkpoint incomplete)
        26         52 db file scattered read
        61        570 db file scattered read
       215        537 db file scattered read
       120        512 db file scattered read
       172        477 db file scattered read
       138         31 db file scattered read
       123        409 db file scattered read
       102        449 db file scattered read
         2      57783 db file parallel write
.....
127 rows selected.
```

我们注意到和 I/O 紧密相关的等待事件这里存在：

db file scattered read 意味着可能存在全表扫描，占用大量的 I/O。

log file switch (checkpoint incomplete) 意味着检查点完成过慢，导致日志无法切换。

而这些因素显然最终都和系统的 I/O 能力直接相关。

9.5.3 进一步的诊断

获知其存储设备为: Sun StorEdge T3 Array

我这样回复：

1．其实根本问题还在于 I/O,检查点不能完成说明 DBWR 的性能太差,写出的过慢。写出慢是因为你的 IO 存在瓶颈,从你的 iostat 信息来看,在 io busy 为 99%左右时,写入才达到 600K 左右,这个太慢了.正常应该有 1~2M 的速度。

2．应该找到全表扫描的语句进行相应优化，减少 I/O 竞争和使用。

3．这样大规模的并发 redo,我觉得是有大批量的数据更改操作导致的,你应该尝试找出这样的 SQL,看是否存在问题,能否优化,从而在根本上解决问题.

可以在问题出现时多做几次 statspack 采样,应该基本可以从报告中发现问题所在。当然,能在当时出现问题时到数据库中抓取 SQL 是最好的.

4．结合另外一个和 Redo 相关的诊断案例

T3 可能存在硬件故障，导致了性能低下，所以也应该检查影响问题。

9.5.4 最后的问题定位

最后这位朋友回复邮件：

您说的很正确，经过这几天的状况比对，以及和另外一台备援机做比对，我认为有两个原因：

1. T3 出了问题，以致于 IO 效率大幅降低，我们试了另外一台「正常」表现的主机，发现做同一动作时，「正常」主机其 IO 写入值可以高到 23000K ，而有问题的 T3 其 IO 值最高写入值仅达 1000K 。已经请硬件厂商查找原因。

2. 我们其中一个表格，竟然已经成长到了近 2000 万笔，原因是某项定期清除数据库之程序出了错误，造成数据量过大，又很不幸的，我们很多查询均是做全表扫瞄，只要碰上处理这个表格的数据，就会造成恶性循环。在后来的 alert log 中，竟然看到了其中数笔查询均花了 2 万五千多秒，还造成了以下问题：

```
ORA-01555 caused by SQL statement below (Query Duration=25889 sec, SCN: 0x0000.21fb3c10):
```

3. 最后发现是硬件问题后，就请硬件厂商检查，发现 T3 的 UPS 电池已经坏了，四个坏了三个，而 UPS 电池会严重影响硬盘写入的效率，从 write back 变成 write through ，所以硬盘效率会低落。更换电池后已经恢复正常。

9.5.5 一点总结

简要的收录这个案例，只是为了给大家提供一个处理问题的思路，从症状入手，结合种种蛛丝马迹，解决数据库问题并非困难。

9.6 总结

Oracle 数据库在管理的过程中，可能会遇到各种各样的问题，解决这些问题，不仅要求我们具备扎实的基础知识，而且要求我们能够把书本中学到的知识灵活运用到实践中去。所以大家在学习的过程中要认真思考，多做实验，把书上的东西切实变成自己的知识，这样才能够在实际工作中得心应手，应付自如。