2018年10月9日

10:58

一波三折：一次CPU使用率过高故障分析SQL优化解决过程

原创： 罗贵林 [数据和云](javascript:void(0);) 昨天

关注我们获得更多内容

**作者 | 罗贵林：** 云和恩墨技术工程师，具有8年以上的 Oracle 数据库工作经验，曾任职于大型的国家电信、省级财政、省级公安的维护，性能调优等。精通 Oracle 数据库管理，调优，问题诊断。擅长 SQL 调优，Oracle Rac 等维护，管理。

**1.问题描述**

2018年9月13日一大早接到客户电话说核心数据库RAC两主机CPU使用很高，90%以上，系统操作缓慢，需要马上紧急处理。

**2.把问题想清楚**

**CPU使用高一般有几种原因？可以从哪些方面入手分析？客户系统情况，故障是高峰期累积的还是突发的？**

* + 一般CPU占用高是由**排序、SQL解析、执行计划突变、全表扫描、会话阻塞**等，可能的原因较多，需要抽丝剥茧，逐步定位根因；
  + 分析方法主要由主机top/topas占CPU高的进程查询相应SQL、会话增长趋势、阻塞分析、ASH/AWR报告分析、SQL执行时间/执行计划变化等；
  + 需要询问客户业务场景，业务上有没有变更，例如开发功能变更、业务使用量增加等，是当前突发的，还是持续了一段时间的问题。

**经询问，本次故障是三天前突发的，问题已持续了三天时间**。

**3.分析：问题逐步排查**

**3.1.  主机topas信息**

本次通过AIX主机topas信息看到进程使用CPU都很平均，无法直接定位是某个进程某个SQL引起的CPU使用过高的问题，**如果可以直接明显看出资源过高损耗，定位可用以下方法：**

**--找到占用系统资源特别大的Oracle的Session及执行的SQL语句(根据进程号查找)**

select a.username,a.machine,a.program,a.sid,a.serial#,a.status,c.piece,c.sql\_text from v$session a, v$process b,v$sqltext c

where b.spid=&pid and b.addr=a.paddr and a.sql\_address=c.address(+) order by c.piece;

**--根据sid查找session的执行语句**

select sid,sql\_text from v$session s,v$sql q

where sid = &sid and (q.sql\_id=s.sql\_id or q.sql\_id = s.prev\_sql\_id);

**3.2.  会话的增长趋势**

通过查询V$SYSMETRIC\_HISTORY视图（记录了上一小时里每1分钟的指标信息，上3分钟里每15s的指标信息），可以得出每分钟内会话数的增长趋势：

select begin\_time,trunc(value)

from V$SYSMETRIC\_HISTORY

where metric\_name='Average Active Sessions'

and group\_id=2 order by begin\_time;

经查询，本次每分钟会话数增长正常，不是会话暴增导致的问题。

**3.3.  等待事件阻情况**

分析ASH信息，查询v$active\_session\_history/ dba\_hist\_active\_sess\_history视图

--等待事件阻情况

SELECT t.SQL\_ID,t.EVENT,COUNT(\*)

from v$active\_session\_history t

WHERE t.SAMPLE\_TIME >= to\_date('2018-09-13 08:30:00', 'yyyy-mm-dd hh24:mi:ss')

and sample\_time < to\_date('2018-09-13 11:00:00', 'yyyy-mm-dd hh24:mi:ss')

GROUP BY t.SQL\_ID,t.EVENT ORDER BY 3 DESC;

--等待事件发生的时间趋势

SELECT TRUNC(sqmple\_time,'mi'),COUNT(1) from v$active\_session\_history t

WHERE t.SAMPLE\_TIME >= to\_date('2018-09-13 08:30:00', 'yyyy-mm-dd hh24:mi:ss')

and sample\_time < to\_date('2018-09-13 11:00:00', 'yyyy-mm-dd hh24:mi:ss')

AND t.EVENT='enq: TX - row lock contention'

GROUP BY trunc(t.SAMPLE\_TIME,'mi')

ORDER BY 1

可以看到行锁等待特别严重，进一步查看等待事件发生的时间趋势：

可以看到在9：15分和9：27分有突发的等待事件发生，但相差不大，需要进一步分析，通过分析等待事件链根源：

with ash as (select /\*+ materialize\*/ \* from   v$active\_session\_history t

 where sample\_time >=

       to\_date('2018-09-13 08:30:00', 'yyyy-mm-dd hh24:mi:ss')

   and sample\_time <

       to\_date('2018-09-13 11:00:00', 'yyyy-mm-dd hh24:mi:ss')),

   chains as (

   select session\_id, level lvl,

          sys\_connect\_by\_path(session\_id||' '|| sql\_id || ' '||event, ' -> ' ) path,

          connect\_by\_isleaf isleaf

     from ash

    start with event in ( 'enq: TX - row lock contention')

    connect by nocycle ( prior blocking\_session = session\_id

    and prior blocking\_session\_serial# = session\_serial# and prior sample\_id = sample\_id))

 select lpad(round (ratio\_to\_report(count(\*)) over () \* 100 )||'%', 5,' ' ) "%This",

        count(\*) samples,

        path

   from chains

  where isleaf = 1

  group by path

  order by samples desc ;

**查看到主要发生行锁的等待SQL：**

-> 2750 85ttdm4hu93yn enq: TX - row lock contention

-> 14518 g78jw4psdfsqc enq: TX - row lock contention

-> 9505 g78jw4psdfsqc enq: TX - row lock contention

-> 13090 0qpy58c2urytn enq: TX - row lock contention

-> 16872 9dmsja7c69xdj enq: TX - row lock contention

-> 505 85ttdm4hu93yn enq: TX - row lock contention

-> 15381 0qpy58c2urytn enq: TX - row lock contention -> 13090 0qpy58c2urytn enq: TX - row lock contention

-> 13756 0qpy58c2urytn enq: TX - row lock contention

-> 13276 85ttdm4hu93yn enq: TX - row lock contention

**但行锁占比都很平均，只能反映出SQL性能较低**，无法与CPU使用率高进行关联，因为很可能是因为CPU高才产生的大量行锁，不是根因而是结果，但也反应出这些行锁的SQL存在性能问题。

**3.4.  AWR整体分析**

业务没变化、top没明显问题、会话正常、阻塞分析无法定位根因，因此进一步取AWR报告进行整体分析（这里可以进行同时段AWR对比分析）。

从9：00到9：30的awr报告中，可以看到，故障时段，853by1q5drtc2语句占用最大的CPU时间和逻辑读：半小时语句执行406次，每次执行19.233秒。明显可以看出是这个SQL引起的CPU使用率过高，接下去就是怎么优化这个SQL的问题。

**4.分析：分析源头是关键**

**4.1.  原因分析**

SELECT  (SELECT IS\_FUND\_VOUCHER

          FROM t1 AVT

         WHERE AV.ADMDIVCODE = AVT.ADMDIVCODE

           AND AV.VT\_CODE = AVT.VT\_CODE) as IS\_FUND\_VOUCHER,

       AV.\*

  FROM t1 AV

 WHERE AV.ADMDIVCODE = 469006

   AND AV.STYEAR = 2018

   AND AV.VT\_CODE = 8202

   AND AV.VOUCHER\_NO = 'v18x010077927'

   AND AV.IS\_CANCEL = 0

   AND AV.IS\_OUT\_DATA = 0

   AND AV.SIGN\_NO =

       (SELECT MAX(SIGN\_NO)

          FROM t1 AV1

         WHERE AV1.ADMDIVCODE = AV.ADMDIVCODE

           AND AV1.STYEAR = AV.STYEAR

           AND AV1.VT\_CODE = AV.VT\_CODE

           AND AV1.VOUCHER\_NO = AV.VOUCHER\_NO

           AND AV1.IS\_CANCEL = 0

           AND EXISTS (SELECT 1

                  FROM t1\_temp AVS1

                 WHERE AVS1.VOUCHER\_ID = AV1.VOUCHER\_ID

                   AND AVS1.STAMP\_STATUS > 0))

通过执行计划可以看到，主要问题在第9行临时表的全表扫描上，被作为驱动表首先进行查询，而临时表的数据量存在不确定性，查看SQL执行计划历史情况：

--查看SQL执行计划历史情况

SELECT s.inst\_id,

s.EXECUTIONS,

ROUND(s.ELAPSED\_TIME/s.EXECUTIONS,2) elapsed\_avg,

ROUND(s.BUFFER\_GETS/s.EXECUTIONS,2) buffer\_avg,

ROUND(s.DISK\_READS/s.EXECUTIONS,2) disk\_avg,

ROUND(s.ROWS\_PROCESSED/s.ELAPSED\_TIME,2) rows\_arg,

s.CHILD\_NUMBER,

s.PLAN\_HASH\_VALUE,

s.SQL\_PROFILE,

s.LAST\_ACTIVE\_TIME,

s.LAST\_LOAD\_TIME

from gv$sql s

WHERE /\*s.SQL\_ID=''

AND \*/s.executions>0;

**可以看到执行计划产生了不合理的变化，执行时间差异很大，这是CPU占用过大的根本原因。**

**4.2.  问题处理**

定位了是SQL问题，首先就要考虑开发是否能修改SQL，如果可以则可以通过改写或指定HINT的方式修正执行计划，实在无法修改SQL的再使用SPM或SQLPROFILE进行绑定，本次开发回复可以修改代码，于是进行如下修改：

SELECT  /\*+ leading(AV) \*/  (SELECT IS\_FUND\_VOUCHER

          FROM t1 AVT

         WHERE AV.ADMDIVCODE = AVT.ADMDIVCODE

           AND AV.VT\_CODE = AVT.VT\_CODE) as IS\_FUND\_VOUCHER,

       AV.\*

  FROM t1 AV

 WHERE AV.ADMDIVCODE = '460200'

   AND AV.STYEAR = 2018

   AND AV.VT\_CODE = '8202'

   AND AV.VOUCHER\_NO = 'HI18x12018082017'

   AND AV.IS\_CANCEL = 0

   AND AV.IS\_OUT\_DATA = 0

   AND AV.SIGN\_NO =

       (SELECT  /\*+ leading(AV1,AVS1) \*/  MAX(SIGN\_NO)

          FROM t1 AV1

         WHERE AV1.ADMDIVCODE = AV.ADMDIVCODE

           AND AV1.STYEAR = AV.STYEAR

           AND AV1.VT\_CODE = AV.VT\_CODE

           AND AV1.VOUCHER\_NO = AV.VOUCHER\_NO

           AND AV1.IS\_CANCEL = 0

           AND EXISTS (SELECT /\*+ INDEX(AVS1,UQ\_TEMP\_2018) \*/ 1

                  FROM t1\_temp AVS1

                 WHERE AVS1.VOUCHER\_ID = AV1.VOUCHER\_ID

                   AND AVS1.STAMP\_STATUS > 0));

指定驱动表后，t1\_temp表使用上了正确的索引，实际执行时间由169秒改变为不到1秒。

**4.3.  处理前后压力对比**

通过开发替换SQL后，可以通过AWR的AAS做简单的前后压力对比，如下示：

从这里可以看到数据库压力**减轻35%左右**。

从主机的TOPAS看到CPU维护在75%左右，本来以为事情到这边就结束了，但客户反馈平时CPU只在50%以下，当前的优化效果不明显，因此通过查看AWR得知加索引的 /\*+ INDEX(AVS1,UQ\_TEMP\_2018) \*/ HINT并没有加到SQL中，和开发再次沟通得知由于索引有以年度作为标识，加索引HINT存在工作量大的困难而没有修改，但却没有通知到我（这里可以看到**和开发确认最终的优化版本是非常重要的**），导致当前优化效果夭折。

**4.4.  承接压力再次分析问题**

**开发无法增加索引HINT，达不到最终的优化效果**，接下去的方法就是绑定执行计划，但这套库的维护是从来没有进行执行计划绑定的，为什么在这里就需要进行绑定？是不是可以不绑定达到效果？

因为SQL已经指定了驱动表，没有引用正确的索引可能存在其他问题。此时如果没有这些思考，就去进行执行计划绑定，就会忽略之后的问题，而只是解决当前问题，多了一些思考后，此时才想到会不会是**统计信息不准**引起的问题，本来是第一时间需要检查的事项，在这次故障处理中被忽略了。

**查看表最后一次分析时间，如下示：**

--查看表最后一次分析时间

SELECT owner,table\_name,last\_analyzed from Dba\_Tab\_Statistics WHERE last\_analyzed IS NOT NULL ORDER BY 3 DESC;

**可以看到很多表统计信息没有自动收集成功，而ORACLE是根据数据改变量大于10%才进行统计信息的收集，且对静态表并不收集统计信息，但当前并不是因为采集任务过重未能及时收集完成，而是基本没有收集。**

--大于10%的改变量的表实际没有及时收集到统计信息，重新收集后dba\_tab\_modifications重新计算

select table\_owner,

       table\_name,

       (inserts + updates + deletes) all\_update\_num,

       (inserts + updates + deletes) /

       nvl((select num\_rows

             from dba\_tables

            where owner = a.table\_owner

              and table\_name = a.table\_name

              and num\_rows <> 0),

           1) \* 100 flag

  from dba\_tab\_modifications a

 where a.table\_owner in ('用户名')

 order by (inserts + updates + deletes) desc;

**能过查询可以确认几乎所有的表都没有进行收集，进一步查询自动统计信息收集信息：**

--查询自动统计信息收集信息

SELECT \* from dba\_autotask\_window\_clients;

SELECT w.window\_name,w.repeat\_interval,w.duration,w.enabled

from Dba\_Autotask\_Window\_Clients c,dba\_scheduler\_windows w

WHERE c.WINDOW\_NAME=w.window\_name AND c.OPTIMIZER\_STATS='ENABLED';

可以看到自动采集任务都停止了，确定统计信息过旧，执行统计信息收集后，

exec dbms\_stats.gather\_table\_stats(ownname =>'用户名' ,tabname => '表名',estimate\_percent =>10,cascade => true);

再次确认853by1q5drtc2语句使用上了正确的执行计划，通过运行execute DBMS\_AUTO\_TASK\_ADMIN.enable();重新开启采集任务。

**至此，CPU使用率高的问题得到了最终解决，最终是由于统计信息没有正确收集导致的，而自动统计信息任务处于关闭状态，**在AWR中853by1q5drtc2语句占用CPU的8.9%，35%的降低可以知道不仅853by1q5drtc2这个SQL因统计信息不准确走错执行计划导致CPU占用高，实际上ASH中那些行锁的SQL也占用了不少的CPU，这里降低效果明显可能是因为这个SQL和行锁语句存在业务关系（这里可以看到优化占用资源消耗第一位的SQL很重要），正确的统计信息收集也将使其他SQL的执行计划更准确。至于为什么采集任务没有开启，由于时间过去久远，已经无法再追踪了。

**5.分析：扩展知识**

这里涉及到统计信息收集的问题，而随着数据量的日益增大，当前库很可能造成自动统计信息不能及时收集完成，因此这里做一下附加调整方法。

**5.1.  锁定部分表使自动统计信息不收集**

如果一些分区表统计信息没有及时收集，可考虑锁定统计信息，采用手工收集

--考虑锁定num\_rows很大，last\_analyzed收集信息较旧的表

select t.table\_owner,

       t.table\_name,

       t.partition\_name,

       t.num\_rows,

       t.last\_analyzed

  from dba\_tab\_partitions t

 where t.table\_owner = 'P2BEMADM'

 order by t.num\_rows desc;

--大于10%的改变量的表实际没有及时收集到统计信息

select table\_owner,

       table\_name,

       (inserts + updates + deletes) all\_update\_num,

       (inserts + updates + deletes) /

       nvl((select num\_rows

             from dba\_tables

            where owner = a.table\_owner

              and table\_name = a.table\_name

              and num\_rows <> 0),

           1) \* 100 flag

  from dba\_tab\_modifications a

 where a.table\_owner in ('P2BEMADM')

 order by (inserts + updates + deletes) desc;

每日的改动量很大，也超过10%，对于这样的表进行锁定统计信息，让自动统计收集过滤这些表，然后定期做手工收集统计信息，放到自动JOB过程里。

dbms\_stats.lock\_table\_stats('用户名','表名');

定期执行以下内容（举例）：

begin

dbms\_stats.unlock\_table\_stats('用户名','表名');

dbms\_stats.gather\_table\_stats(ownname => '用户名',tabname =>’表名’,estimate\_percent => 0.2,cascade =>true);

dbms\_stats.gather\_table\_stats(ownname => '用户名',tabname =>’表名’,partname =>’分区名’,estimate\_percent => 0.2,cascade =>true);

dbms\_stats.gather\_table\_stats(ownname=>'用户名',tabname=>'表名',granularty=>'APPROX\_GLOBAL AND PARTITION',partname=>'分区名',estimate=>0.00001,cascade=>true);

dbms\_stats.lock\_table\_stats('用户名','表名');

end;

5.2.  临时表禁用自动统计信息收集\_参考方案

如果存在临时表（global temporary table），为避免数据库引用错误的临时表统计信息，禁用临时表统计信息收集，如下示：

禁用临时表统计信息收集

select s.table\_name, s.STATTYPE\_LOCKED

     from dba\_TAB\_STATISTICS s

    where S.table\_name IN (select T1.table\_name

                          from dba\_tables t1

                         where t1.temporary = 'Y'

                           and owner = '用户名')

其中STATTYPE\_LOCKED字段为空表示没有锁定统计信息。

为避免数据库引用错误的临时表统计信息进行锁定临时表统计信息，而让其进行动态采样收集

begin

dbms\_stats.unlock\_table\_stats('用户名','表名');

dbms\_stats.delete\_table\_stats(用户名,'表名');

dbms\_stats.lock\_table\_stats(用户名,'表名');

end;

**5.3.  启用11G新特性的增量统计信息收集技术**

除锁定表不自动收集统计信息外，也可考虑使用11G新特性的增量统计信息收集技术，**当系统有很大的分区表时，如果总是全部收集则会比较慢，11g之后可以设置INCREMENTAL只对数据有变动的分区做收集，如下示：**

exec dbms\_stats.set\_table\_prefs('JD','IMS\_RES\_MONITOR\_2','INCREMENTAL','TRUE');

exec dbms\_stats.set\_table\_prefs(user,'table\_name','INCREMENTAL','TRUE'); --只收集数据变动的分区

select dbms\_stats.get\_prefs('INCREMENTAL',null,'table\_name') from dual;  --查看分区表INCREMENTAL的值

注：启用增量统计信息收集会占用SYSAUX表空间，在设定前需要加大SYSAUX表空间大小。

**5.4.  延长自动统计信息收集时间**

自动收集统计信息的时间是周一到周五每晚22:00:00到第二天2:00结束，周六周日两天全天收集。在经过以上方法优化统计信息收集后，可进一步考虑工作日时段延长2小时，即周一到周五时段统计信息收集到第二天4:00结束。

**修改脚本如下：**

--下面修改配置(使用sys用户登录oracle执行)

sqlplus sys/xxx@xxxdb as sysdba

--修改SATURDAY\_WINDOW、SUNDAY\_WINDOW的配置 (改成和平常相同,即每日都是22:00向后4小时,至次日凌晨2点)

--下面的方式也可修改周六、周日的时间Window

begin

  sys.dbms\_scheduler.set\_attribute(name => 'SYS.SATURDAY\_WINDOW', attribute => 'repeat\_interval', value => 'Freq=daily;ByDay=SAT;ByHour=22;ByMinute=0;BySecond=0');

  sys.dbms\_scheduler.set\_attribute(name => 'SYS.SATURDAY\_WINDOW', attribute => 'duration', value => '0 04:00:00');

end;

/

begin

  sys.dbms\_scheduler.set\_attribute(name => 'SYS.SUNDAY\_WINDOW', attribute => 'repeat\_interval', value => 'Freq=daily;ByDay=SUN;ByHour=22;ByMinute=0;BySecond=0');

  sys.dbms\_scheduler.set\_attribute(name => 'SYS.SUNDAY\_WINDOW', attribute => 'duration', value => '0 04:00:00');

end;

/

--查看修改结果:

select t1.window\_name,t1.repeat\_interval,t1.duration from dba\_scheduler\_windows t1,dba\_scheduler\_wingroup\_members t2

 where t1.window\_name=t2.window\_name and t2.window\_group\_name='MAINTENANCE\_WINDOW\_GROUP';

/\*           WINDOW\_NAME        REPEAT\_INTERVAL        DURATION

1        MONDAY\_WINDOW        freq=daily;byday=MON;byhour=22;byminute=0; bysecond=0        +000 04:00:00

2        TUESDAY\_WINDOW        freq=daily;byday=TUE;byhour=22;byminute=0; bysecond=0        +000 04:00:00

3        WEDNESDAY\_WINDOW        freq=daily;byday=WED;byhour=22;byminute=0; bysecond=0        +000 04:00:00

4        THURSDAY\_WINDOW        freq=daily;byday=THU;byhour=22;byminute=0; bysecond=0        +000 04:00:00

5        FRIDAY\_WINDOW        freq=daily;byday=FRI;byhour=22;byminute=0; bysecond=0        +000 04:00:00

6        SATURDAY\_WINDOW        Freq=daily;ByDay=SAT;ByHour=22;ByMinute=0;BySecond=0        +000 04:00:00

7        SUNDAY\_WINDOW        Freq=daily;ByDay=SUN;ByHour=22;ByMinute=0;BySecond=0        +000 04:00:00

\*/

**6.总结：思路决定效率**

总结这个故障，究其原因就是**一次统计信息失真**导致SQL执行计划走错的简单案例，但有时往往入手的思路错了，导致中间分析的道路过于曲折，且与客户的实际沟通与诉求存在偏差（以为加了HINT使SQL达到优化效果就结束了），往往导致问题处理的曲折性。

**在实践中，针对故障和问题需要充分考虑多个可能性，以其一击中的**，在本次的故障处理中最终分析是由于统计信息未自动收集导致一系列的性能问题。

**总结SQL执行计划走错的问题注意以下几点：**

1. SQL执行计划是不是突变的，这个通过SQL执行历史信息可以查看到。

2. SQL涉及的表的统计信息是不是最新的，如果不是，需要重新收集。

3. 在开发可以变更的情况下，不建议使用绑定执行计划的方法，使用通用HINT方法优于执行计划绑定。

4. 开发无法变更的情况下，可以通过绑定执行计划的方法快速处理SQL执行效率低下的问题。

另外，还需要及时熟悉客户现场情况和实际变更情况，避免沟通确认上引起一些不必要的麻烦和误会，延误分析时间和实际优化效果，故障处理后，还需要评估后续可能产生的问题，予以准备，在本次故障处理中收集了一些自动统计信息的优化方法以备后续使用。

源文档 <<https://mp.weixin.qq.com/s/e4ZNy7o6_aoGmBt7sVs6MA>>