# 如何通过dba\_hist\_active\_sess\_history分析数据库历史性能问题

By: [Shunlun Pan](https://blogs.oracle.com/database4cn/90ea0298-5263-420d-8bc8-ec98fb4928fc)

**如何通过dba\_hist\_active\_sess\_history分析数据库历史性能问题**

**背景  
在很多情况下，当数据库发生性能问题的时候，我们并没有机会来收集足够的诊断信息，比如system state dump或者hang analyze，甚至问题发生的时候DBA根本不在场。这给我们诊断问题带来很大的困难。那么在这种情况下，我们是否能在事后收集一些信息来分析问题的原因呢？在Oracle 10G或者更高版本上，答案是肯定的。本文我们将介绍一种通过dba\_hist\_active\_sess\_history的数据来分析问题的一种方法。**

**适用于  
Oracle 10G或更高版本，本文适用于任何平台。**

**详情  
在Oracle 10G中，我们引入了AWR和ASH采样机制，有一个视图gv$active\_session\_history会每秒钟将数据库所有节点的Active Session采样一次，而dba\_hist\_active\_sess\_history则会将gv$active\_session\_history里的数据每10秒采样一次并持久化保存。基于这个特征，我们可以通过分析dba\_hist\_active\_sess\_history的Session采样情况，来定位问题发生的准确时间范围，并且可以观察每个采样点的top event和top holder。下面通过一个例子来详细说明。**

**1. Dump出问题期间的ASH数据：  
为了不影响生产系统，我们可以将问题大概期间的ASH数据export出来在测试机上分析。  
 基于dba\_hist\_active\_sess\_history创建一个新表m\_ash，然后将其通过exp/imp导入到测试机。在发生问题的数据库上执行exp:  
 SQL> conn user/passwd  
 SQL> create table m\_ash as select \* from dba\_hist\_active\_sess\_history where SAMPLE\_TIME between TO\_TIMESTAMP ('<time\_begin>', 'YYYY-MM-DD HH24:MI:SS') and TO\_TIMESTAMP ('<time\_end>', 'YYYY-MM-DD HH24:MI:SS');**

$ exp user/passwd file=m\_ash.dmp tables=(m\_ash) log=m\_ash.exp.log

然后导入到测试机：  
 $ imp user/passwd file=m\_ash.dmp log=m\_ash.imp.log

**2. 验证导出的ASH时间范围:  
为了加快速度，我们采用了并行查询。另外建议采用Oracle SQL Developer来查询以防止输出结果折行不便于观察。**

set line 200 pages 1000  
 col sample\_time for a25  
 col event for a40  
 alter session set nls\_timestamp\_format='yyyy-mm-dd hh24:mi:ss.ff';

select /\*+ parallel 8 \*/  
  t.dbid, t.instance\_number, min(sample\_time), max(sample\_time), count(\*) session\_count  
   from m\_ash t  
  group by t.dbid, t.instance\_number  
  order by dbid, instance\_number;

INSTANCE\_NUMBER    MIN(SAMPLE\_TIME)    MAX(SAMPLE\_TIME)    SESSION\_COUNT  
 1    2015-03-26 21:00:04.278    2015-03-26 22:59:48.387    2171  
 2    2015-03-26 21:02:12.047    2015-03-26 22:59:42.584    36

从以上输出可知该数据库共2个节点，采样时间共2小时，节点1的采样比节点2要多很多，问题可能发生在节点1上。

**3. 确认问题发生的精确时间范围:  
参考如下脚本：**

select /\*+ parallel 8 \*/  
  dbid, instance\_number, sample\_id, sample\_time, count(\*) session\_count  
   from m\_ash t  
  group by dbid, instance\_number, sample\_id, sample\_time  
  order by dbid, instance\_number, sample\_time;

INSTANCE\_NUMBER    SAMPLE\_ID    SAMPLE\_TIME    SESSION\_COUNT  
 1    36402900    2015-03-26 22:02:50.985    4  
 1    36402910    2015-03-26 22:03:01.095    1  
 1    36402920    2015-03-26 22:03:11.195    1  
 1    36402930    2015-03-26 22:03:21.966    21  
 1    36402940    2015-03-26 22:03:32.116    102  
 1    36402950    2015-03-26 22:03:42.226    181  
 1    36402960    2015-03-26 22:03:52.326    200  
 1    36402970    2015-03-26 22:04:02.446    227  
 1    36402980    2015-03-26 22:04:12.566    242  
 1    36402990    2015-03-26 22:04:22.666    259  
 1    36403000    2015-03-26 22:04:32.846    289  
 1    36403010    2015-03-26 22:04:42.966    147  
 1    36403020    2015-03-26 22:04:53.076    2  
 1    36403030    2015-03-26 22:05:03.186    4  
 1    36403040    2015-03-26 22:05:13.296    1  
 1    36403050    2015-03-26 22:05:23.398    1

注意观察以上输出的每个采样点的active session的数量，数量突然变多往往意味着问题发生了。从以上输出可以确定问题发生的精确时间在2015-03-26 22:03:21 ~ 22:04:42，问题持续了大约1.5分钟。  
 注意: 观察以上的输出有无断档，比如某些时间没有采样。

**4. 确定每个采样点的top n event:  
在这里我们指定的是top 2 event 并且注掉了采样时间以观察所有采样点的情况。如果数据量较多，您也可以通过开启sample\_time的注释来观察某个时间段的情况。注意最后一列session\_count指的是该采样点上的等待该event的session数量。**

select t.dbid,  
        t.sample\_id,  
        t.sample\_time,  
        t.instance\_number,  
        t.event,  
        t.session\_state,  
        t.c session\_count  
   from (select t.\*,  
                rank() over(partition by dbid, instance\_number, sample\_time order by c desc) r  
           from (select /\*+ parallel 8 \*/  
                  t.\*,  
                  count(\*) over(partition by dbid, instance\_number, sample\_time, event) c,  
                  row\_number() over(partition by dbid, instance\_number, sample\_time, event order by 1) r1  
                   from m\_ash t  
                 /\*where sample\_time >  
                     to\_timestamp('2013-11-17 13:59:00',  
                                  'yyyy-mm-dd hh24:mi:ss')  
                 and sample\_time <  
                     to\_timestamp('2013-11-17 14:10:00',  
                                  'yyyy-mm-dd hh24:mi:ss')\*/  
                 ) t  
          where r1 = 1) t  
  where r < 3  
  order by dbid, instance\_number, sample\_time, r;

SAMPLE\_ID    SAMPLE\_TIME    INSTANCE\_NUMBER    EVENT    SESSION\_STATE    SESSION\_COUNT  
 36402900    22:02:50.985    1        ON CPU    3  
 36402900    22:02:50.985    1    db file sequential read    WAITING    1  
 36402910    22:03:01.095    1        ON CPU    1  
 36402920    22:03:11.195    1    db file parallel read    WAITING    1  
 36402930    22:03:21.966    1    cursor: pin S wait on X    WAITING    11  
 36402930    22:03:21.966    1    latch: shared pool    WAITING    4  
 36402940    22:03:32.116    1    cursor: pin S wait on X    WAITING    83  
 36402940    22:03:32.116    1    SGA: allocation forcing component growth    WAITING    16  
 36402950    22:03:42.226    1    cursor: pin S wait on X    WAITING    161  
 36402950    22:03:42.226    1    SGA: allocation forcing component growth    WAITING    17  
 36402960    22:03:52.326    1    cursor: pin S wait on X    WAITING    177  
 36402960    22:03:52.326    1    SGA: allocation forcing component growth    WAITING    20  
 36402970    22:04:02.446    1    cursor: pin S wait on X    WAITING    204  
 36402970    22:04:02.446    1    SGA: allocation forcing component growth    WAITING    20  
 36402980    22:04:12.566    1    cursor: pin S wait on X    WAITING    219  
 36402980    22:04:12.566    1    SGA: allocation forcing component growth    WAITING    20  
 36402990    22:04:22.666    1    cursor: pin S wait on X    WAITING    236  
 36402990    22:04:22.666    1    SGA: allocation forcing component growth    WAITING    20  
 36403000    22:04:32.846    1    cursor: pin S wait on X    WAITING    265  
 36403000    22:04:32.846    1    SGA: allocation forcing component growth    WAITING    20  
 36403010    22:04:42.966    1    enq: US - contention    WAITING    69  
 36403010    22:04:42.966    1    latch: row cache objects    WAITING    56  
 36403020    22:04:53.076    1    db file scattered read    WAITING    1  
 36403020    22:04:53.076    1    db file sequential read    WAITING    1

从以上输出我们可以发现问题期间最严重的等待为cursor: pin S wait on X，高峰期等待该event的session数达到了265个，其次为SGA: allocation forcing component growth，高峰期session为20个。

注意:  
 1) 再次确认以上输出有无断档，是否有某些时间没有采样。  
 2) 注意那些session\_state为ON CPU的输出，比较ON CPU的进程个数与您的OS物理CPU的个数，如果接近或者超过物理CPU个数，那么您还需要检查OS当时的CPU资源状况，比如OSWatcher/NMON等工具，高的CPU Run Queue可能引发该问题，当然也可能是问题的结果，需要结合OSWatcher和ASH的时间顺序来验证。

**5. 观察每个采样点的等待链:  
其原理为通过dba\_hist\_active\_sess\_history. blocking\_session记录的holder来通过connect by级联查询，找出最终的holder. 在RAC环境中，每个节点的ASH采样的时间很多情况下并不是一致的，因此您可以通过将本SQL的第二段注释的sample\_time稍作修改让不同节点相差1秒的采样时间可以比较(注意最好也将partition by中的sample\_time做相应修改)。该输出中isleaf=1的都是最终holder，而iscycle=1的代表死锁了(也就是在同一个采样点中a等b，b等c，而c又等a，这种情况如果持续发生，那么尤其值得关注)。采用如下查询能观察到阻塞链。**

select /\*+ parallel 8 \*/  
  level                     lv,  
  connect\_by\_isleaf         isleaf,  
  connect\_by\_iscycle        iscycle,  
  t.dbid,  
  t.sample\_id,  
  t.sample\_time,  
  t.instance\_number,  
  t.session\_id,  
  t.sql\_id,  
  t.session\_type,  
  t.event,  
  t.session\_state,  
  t.blocking\_inst\_id,  
  t.blocking\_session,  
  t.blocking\_session\_status  
   from m\_ash t  
 /\*where sample\_time >  
     to\_timestamp('2013-11-17 13:55:00',  
                  'yyyy-mm-dd hh24:mi:ss')  
 and sample\_time <  
     to\_timestamp('2013-11-17 14:10:00',  
                  'yyyy-mm-dd hh24:mi:ss')\*/  
  start with blocking\_session is not null  
 connect by nocycle  
  prior dbid = dbid  
        and prior sample\_time = sample\_time  
           /\*and ((prior sample\_time) - sample\_time between interval '-1'  
           second and interval '1' second)\*/  
        and prior blocking\_inst\_id = instance\_number  
        and prior blocking\_session = session\_id  
        and prior blocking\_session\_serial# = session\_serial#  
  order siblings by dbid, sample\_time;

LV    ISLEAF    ISCYCLE    SAMPLE\_TIME    INSTANCE\_NUMBER    SESSION\_ID    SQL\_ID    EVENT    SESSION\_STATE    BLOCKING\_INST\_ID    BLOCKING\_SESSION    BLOCKING\_SESSION\_STATUS  
 1    0    0    22:04:32.846    1    1259    3ajt2htrmb83y    cursor:    WAITING    1    537    VALID  
 2    1    0    22:04:32.846    1    537    3ajt2htrmb83y    SGA:    WAITING            UNKNOWN

注意为了输出便于阅读，我们将等待event做了简写，下同。从上面的输出可见，在相同的采样点上(22:04:32.846)，节点1 session 1259在等待cursor: pin S wait on X，其被节点1 session 537阻塞，而节点1 session 537又在等待SGA: allocation forcing component growth，并且ASH没有采集到其holder，因此这里cursor: pin S wait on X只是一个表面现象，问题的原因在于SGA: allocation forcing component growth

**6. 基于第5步的原理来找出每个采样点的最终top holder:  
比如如下SQL列出了每个采样点top 2的blocker session，并且计算了其最终阻塞的session数(参考blocking\_session\_count)**

select t.lv,  
        t.iscycle,  
        t.dbid,  
        t.sample\_id,  
        t.sample\_time,  
        t.instance\_number,  
        t.session\_id,  
        t.sql\_id,  
        t.session\_type,  
        t.event,  
        t.seq#,  
        t.session\_state,  
        t.blocking\_inst\_id,  
        t.blocking\_session,  
        t.blocking\_session\_status,  
        t.c blocking\_session\_count  
   from (select t.\*,  
                row\_number() over(partition by dbid, instance\_number, sample\_time order by c desc) r  
           from (select t.\*,  
                        count(\*) over(partition by dbid, instance\_number, sample\_time, session\_id) c,  
                        row\_number() over(partition by dbid, instance\_number, sample\_time, session\_id order by 1) r1  
                   from (select /\*+ parallel 8 \*/  
                          level              lv,  
                          connect\_by\_isleaf  isleaf,  
                          connect\_by\_iscycle iscycle,  
                          t.\*  
                           from m\_ash t  
                         /\*where sample\_time >  
                             to\_timestamp('2013-11-17 13:55:00',  
                                          'yyyy-mm-dd hh24:mi:ss')  
                         and sample\_time <  
                             to\_timestamp('2013-11-17 14:10:00',  
                                          'yyyy-mm-dd hh24:mi:ss')\*/  
                          start with blocking\_session is not null  
                         connect by nocycle  
                          prior dbid = dbid  
                                and prior sample\_time = sample\_time  
                                   /\*and ((prior sample\_time) - sample\_time between interval '-1'  
                                   second and interval '1' second)\*/  
                                and prior blocking\_inst\_id = instance\_number  
                                and prior blocking\_session = session\_id  
                                and prior  
                                     blocking\_session\_serial# = session\_serial#) t  
                  where t.isleaf = 1) t  
          where r1 = 1) t  
  where r < 3  
  order by dbid, sample\_time, r;

SAMPLE\_TIME    INSTANCE\_NUMBER    SESSION\_ID    SQL\_ID    EVENT    SEQ#    SESSION\_STATE    BLOCKING\_SESSION\_STATUS    BLOCKING\_SESSION\_COUNT  
 22:03:32.116    1    1136    1p4vyw2jan43d    SGA:    1140    WAITING    UNKNOWN    82  
 22:03:32.116    1    413    9g51p4bt1n7kz    SGA:    7646    WAITING    UNKNOWN    2  
 22:03:42.226    1    1136    1p4vyw2jan43d    SGA:    1645    WAITING    UNKNOWN    154  
 22:03:42.226    1    537    3ajt2htrmb83y    SGA:    48412    WAITING    UNKNOWN    4  
 22:03:52.326    1    1136    1p4vyw2jan43d    SGA:    2150    WAITING    UNKNOWN    165  
 22:03:52.326    1    537    3ajt2htrmb83y    SGA:    48917    WAITING    UNKNOWN    8  
 22:04:02.446    1    1136    1p4vyw2jan43d    SGA:    2656    WAITING    UNKNOWN    184  
 22:04:02.446    1    537    3ajt2htrmb83y    SGA:    49423    WAITING    UNKNOWN    10  
 22:04:12.566    1    1136    1p4vyw2jan43d    SGA:    3162    WAITING    UNKNOWN    187  
 22:04:12.566    1    2472        SGA:    1421    WAITING    UNKNOWN    15  
 22:04:22.666    1    1136    1p4vyw2jan43d    SGA:    3667    WAITING    UNKNOWN    193  
 22:04:22.666    1    2472        SGA:    1926    WAITING    UNKNOWN    25  
 22:04:32.846    1    1136    1p4vyw2jan43d    SGA:    4176    WAITING    UNKNOWN    196  
 22:04:32.846    1    2472        SGA:    2434    WAITING    UNKNOWN    48

注意以上输出，比如第一行，代表在22:03:32.116，节点1的session 1136最终阻塞了82个session.  顺着时间往下看，可见节点1的session 1136是问题期间最严重的holder，它在每个采样点都阻塞了100多个session，并且它持续等待SGA: allocation forcing component growth，注意观察其seq#您会发现该event的seq#在不断变化，表明该session并未完全hang住，由于时间正好发生在夜间22:00左右,这显然是由于自动收集统计信息job导致shared memory resize造成，因此可以结合Scheduler/MMAN的trace以及dba\_hist\_memory\_resize\_ops的输出进一步确定问题。

注意：  
 1) blocking\_session\_count 指某一个holder最终阻塞的session数，比如 a <- b<- c （a被b阻塞,b又被c阻塞），只计算c阻塞了1个session，因为中间的b可能在不同的阻塞链中发生重复。  
 2) 如果最终的holder没有被ash采样(一般因为该holder处于空闲)，比如 a<- c 并且b<- c (a被c阻塞，并且b也被c阻塞)，但是c没有采样，那么以上脚本无法将c统计到最终holder里，这可能会导致一些遗漏。  
 3) 注意比较blocking\_session\_count的数量与第3步查询的每个采样点的总session\_count数，如果每个采样点的blocking\_session\_count数远小于总session\_count数，那表明大部分session并未记载holder，因此本查询的结果并不能代表什么。  
 4) 在Oracle 10g中，ASH并没有blocking\_inst\_id列，在以上所有的脚本中，您只需要去掉该列即可，因此10g的ASH一般只能用于诊断单节点的问题。

**其他关于ASH的应用  
除了通过ASH数据来找holder以外，我们还能用它来获取很多信息(基于数据库版本有所不同)：  
 比如通过PGA\_ALLOCATED列来计算每个采样点的最大PGA，合计PGA以分析ora-4030/Memory Swap相关问题；  
 通过TEMP\_SPACE\_ALLOCATED列来分析临时表空间使用情况；  
 通过IN\_PARSE/IN\_HARD\_PARSE/IN\_SQL\_EXECUTION列来分析SQL处于parse还是执行阶段;  
 通过CURRENT\_OBJ#/CURRENT\_FILE#/CURRENT\_BLOCK#来确定I/O相关等待发生的对象.**

**Join the discussion**