DTCC

2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

利用DTRACE定位Oracle高并发堵塞案例

栾长苗

主要内容

- 什么是DTrace
- 为什么使用DTrace
- 如何使用DTrace
- Dtrace定位Oracle高并发案例

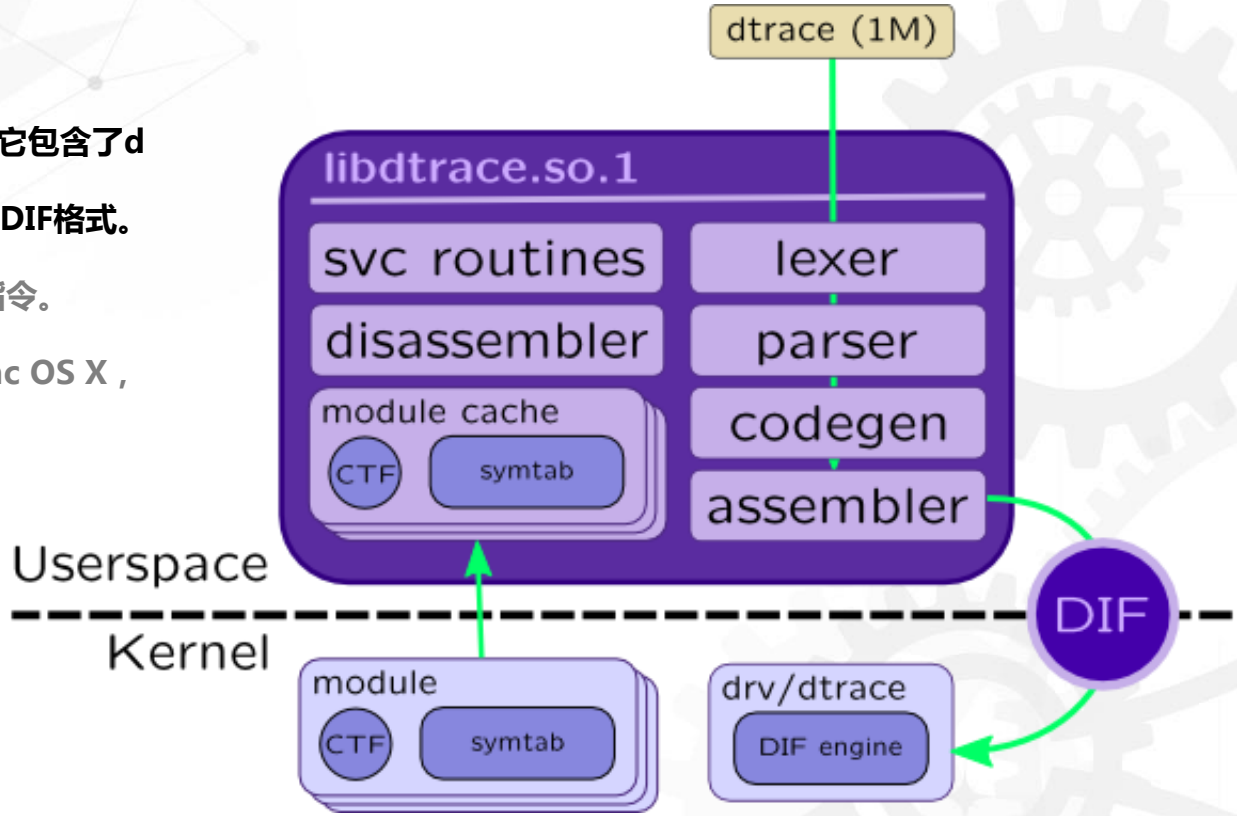
什么是DTrace

- Dtrace是一个动态跟踪工具
- 用来在生产和试验性生产系统上找出系统瓶颈的工具
- 可以通过D脚本语言创建定制程序
- 在系统中插入大量（大概7万个）的probe，然后通过Dtrace激活这些探测器。
- 记录和显示与内核或用户进程的相关信息，包括参数变量，调用时间，调用次数等等。
- 跟踪堆栈，可以指明函数调用的代码。

```
183    | kcc_scan:entry                    -> entry of kcc_scan
183     -> kccgftcs                        -> entry of kccgftcs
183       -> kccrof                        -> entry of kccrof
183         <- ksdpec                      9840 nsec on ksdpec
......
183           -> kccrec_rbl                -> entry of kccrec_rbl
183             -> kccbmp_get              -> entry of kccbmp_get
183               -> kslwtb_resmgr         -> entry of kslwtb_resmgr
183                 -> kskthbwt            -> entry of kskthbwt
183                 <- kskthbwt            7430 nsec on kskthbwt
......
183                   -> ksdpec            -> entry of ksdpec
183                   <- ksdpec            5820 nsec on ksdpec
.....
183                   -> pread             -> entry of pread
183                    | pread:entry       read 655360 bytes from 22691840 pos
on the file 256
183                   <- pread             2171790 nsec on pread
......
183                 <- kslwte_tm           28630 nsec on kslwte_tm
183               <- kslwte_resmgr         39310 nsec on kslwte_resmgr
183             <- kccgft_prefetch         2384290 nsec on kccgft_prefetch
183             -> kccpb_sanity_check      -> entry of kccpb_sanity_check
183               -> ksdpec                -> entry of ksdpec
183               <- ksdpec                5190 nsec on ksdpec
183             <- kccpb_sanity_check      15720 nsec on kccpb_sanity_check
183           <- kccrec_rbl                2448100 nsec on kccrec_rbl
183         <- kccext_ugg                  27824950 nsec on kccext_ugg
183       <- kccrec_read_write             28206320 nsec on kccrec_read_write
183       -> kcc_rno_to_rid                -> entry of kcc_rno_to_rid
183       <- kcc_rno_to_rid                8900 nsec on kcc_rno_to_rid
183     <- kccgftcs                        29635100 nsec on kccgftcs
......
183     <- kccdebug_set                    6450 nsec on kccdebug_set
183    | kcc_scan:return                   29690700 nsec on kcc_scan
183   <- kcc_scan                          29710330 nsec on this function
```

什么是DTrace

- DTrace不需要任何内核代码修改。
- Dtrace的核心是libdtrace.so.1 library，它包含了d脚本的编译器。该编译器将D脚本编译成了DIF格式。
- Dtrace驱动可以解释执行DIF格式的机器指令。
- 支持多个操作系统Solaris，Freebsd，Mac OS X，Linux（BPF）



为什么使用DTrace

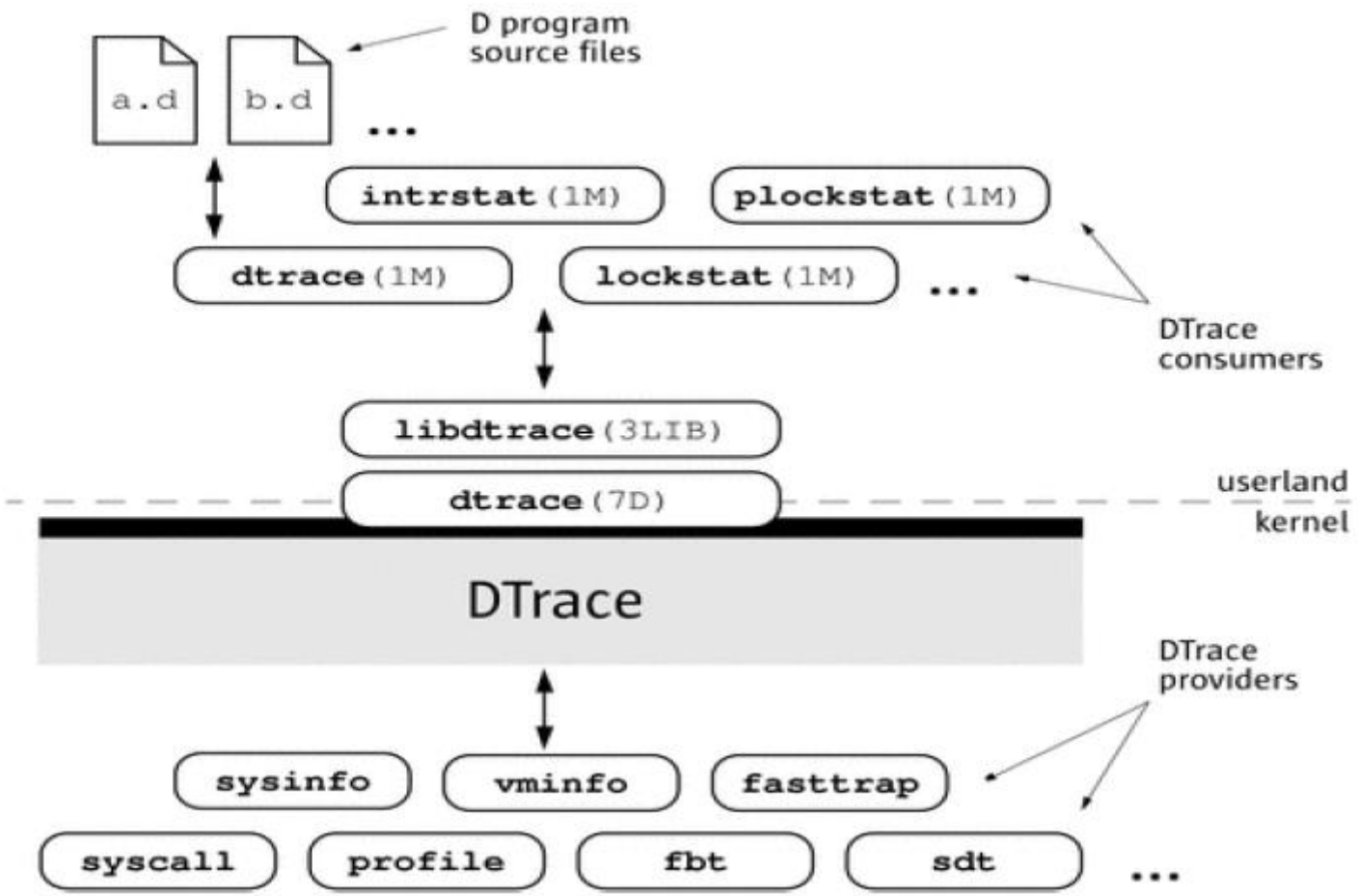
和truss相比：

- **Dtrace更安全（代码被严格限制），**
- **更少的资源**
- **更准确**
- **异步调用**

业务变化

- **活动多**
- **高并发**
- **问题持续时间短，不易重现**
- **故障汇报-问题根源**
- **问题整改**

如何使用DTrace


D program
source files
a.d
b.d
...
intrstat(1M)
plockstat(1M)
dtrace(1M)
lockstat(1M)
...
DTrace
consumers
libdtrace(3LIB)
dtrace(7D)
userland
kernel
DTrace
DTrace
providers
sysinfo
vminfo
fasttrap
syscall
profile
fbt
sdt
...

如何使用DTrace

- D脚本文件格式

```
# more /tmp/syscall.d
BEGIN
{  cnt = 0; } --初始化定义变量

syscall::pread:entry   --probe说明（在哪里追踪），格式：探针:模块:函数:entry。
/ pid == 8333 && arg0 == 256 /  --predict（什么时候执行），条件判断为真。
{
   ustack(50,10); --action，具体做什么操作。这儿是ustack跟踪用户线程的栈。
   cnt++;
}

END
{  printf("%3d", cnt); }
```

- 使用dtrace调用D脚本

```
# dtrace -s /tmp/syscall.d
dtrace: script '/tmp/syscall.d' matched 3 probes
CPU     ID                    FUNCTION:NAME
 65   5140                      pread:entry
              libc.so.1`_pread+0x8
              oracle`skgfqio+0x204
              oracle`opimai_real+0x10c
              oracle`main+0x98
              oracle`_start+0x17c
```

Dtrace定位Oracle高并发案例

- 案例描述
- 案例解析
- 案例总结
- 案例整改

案例描述

从11月10号到11月19日的日志查看，两次出现中间件连接池suspend是在11-11和11-12 日，同时应用有持续1 ~ 2分钟缓慢响应。

11号suspended的app:：
lifeSF1203 10.33.98.13
lifeSF7131 10.33.98.11
lifeSF7143 10.33.98.11
lifeSF7308 10.33.98.13

```
java.sql.SQLRecoverableException: IO 错误: The Network Adapter could not establish the connection
        at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:458)
        at oracle.jdbc.driver.PhysicalConnection.<init>(PhysicalConnection.java:546)
```

12号suspended 的app:
lifeSF 7131 10.33.98.11
lifeSF 7132 10.33.98.12
lifeSF 7143 10.33.98.11

19号suspended 的app，并出现Java OOM错误:
lifeSF 7131 10.33.98.11
lifeSF 7132 10.33.98.12

案例分析

- **中间件问题**
- **业务有连接风暴**
- **数据库出现堵塞**

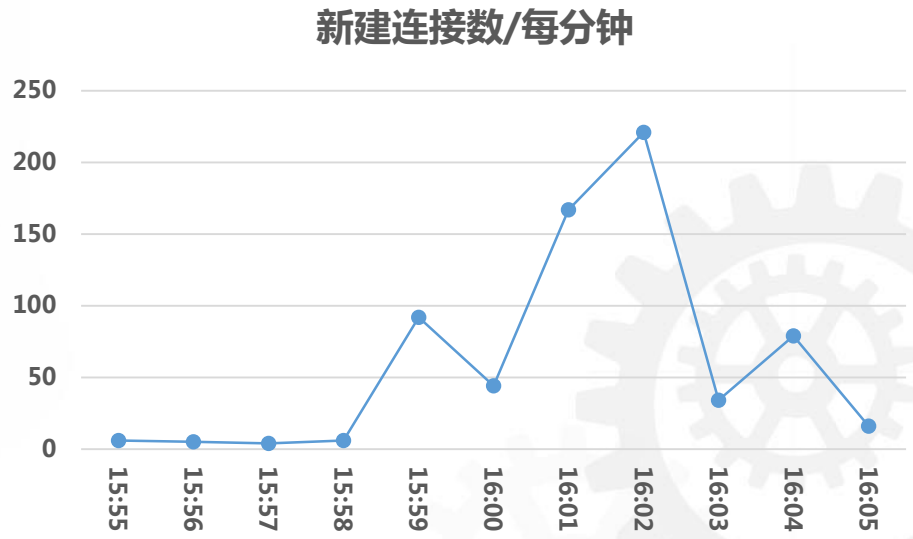
中间件连接泄漏

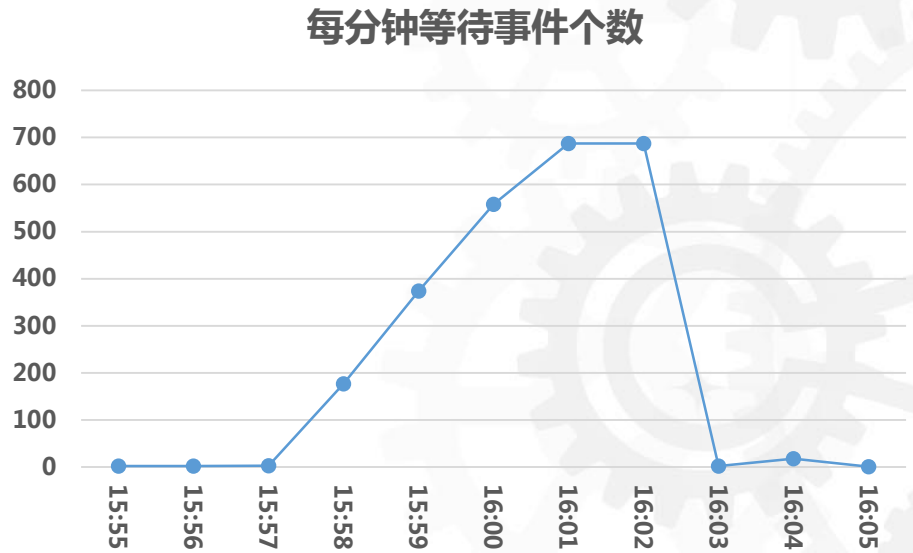
有业务活动，导致连接风暴：抢售，直播

数据库出现异常，有性能瓶颈。响应变慢，反过来导致应用疯狂连接进来。

案例分析

- 监听处理遇到瓶颈
- 业务是否有连接风暴
- 数据库出现堵塞（ log file sync ）





案例分析

- Log file sync会话 - > Lgwr进程（enq: CF – contention） - > ARC6 进程（control file sequential read）

1. Lgwr进程是处于enq: CF – contention，其blocking session 是1629，是arc6进程

SAMPLE_ID	PROGRAM	EVENT	BLOCKING_SESSION	COUNT(*)
133207303	oracle@g4as3020 (LGWR)	enq: CF – contention	1629	1
133207304	oracle@g4as3020 (LGWR)	enq: CF – contention	1629	1
133207305	oracle@g4as3020 (LGWR)	enq: CF – contention	1629	1
133207306	oracle@g4as3020 (LGWR)	enq: CF – contention	1629	1

2. arc6进程处于control file sequential read。

SAMPLE_ID	PROGRAM	EVENT	BLOCKING_SESSION	COUNT(*)
133207303	oracle@g4as3020 (ARC6)	control file sequential read		1
133207304	oracle@g4as3020 (ARC6)	control file sequential read		1
133207305	oracle@g4as3020 (ARC6)	control file sequential read		1
133207306	oracle@g4as3020 (ARC6)	control file sequential read		1

- IO慢
- Bug

类似bug分析

在MOS中， 虽然可以看到有类似的bug，如bug 353351导致arch进程is looping holding CF enqueue，如bug 4505208，Arch process is CF enqueuee holder， 但是这些bug已经在10.2.以后修复。

进一步检查发现，在DBXS1库中 有隐含参数 _log_archive_callout='LOCAL_FIRST=TRUE' 的设置，但是这个参数在10g中已经过期，在10g中已经被log_archive_local_first替 代。

在之前bug的 检查中，也看到由于FAL检测的缘故，造成arch process hold住CF enqueue。因此，建议取消该隐含参数，直接使用log_archive_local_first为true（默认值），后续再继续观察。

ARC6归档进程

尝试对arc6进程进行truss：

```
17030/1:        64.6171  0.0009  0.0001 pread(256, "15C2\0\0\0\0\0 - 0C8 A 3".., 16384, 737280) = 16384
17030/1:        64.6183  0.0012  0.0001 stat("/xxx/xxx/dbxs1/redo/dbxs1/redo08.log", 0xFFFFFFFF7FFF9460) = 0
17030/1:        66.5604  1.9421  0.0001 open("/xxx/xxx/dbxs1/redo/dbxs1/redo08.log", O_RDONLY) = 27
17030/1:        68.5034  1.9430  0.0000 fstatvfs(27, 0xFFFFFFFF7FFF8C78)              = 0
17030/1:        68.5036  0.0002  0.0000 fstatvfs(27, 0xFFFFFFFF7FFF93D8)              = 0
17030/1:        68.5050  0.0014  0.0000 close(27)                                    = 0
17030/1:        68.5053  0.0003  0.0001 open("/xxx/xxx/dbxs1/redo/dbxs1/redo08.log", O_RDWR|O_DSYNC) = 27
17030/1:        70.4453  1.9400  0.0000 getrlimit(RLIMIT_NOFILE, 0xFFFFFFFF7FFF9418)    = 0
17030/1:        70.4455  0.0002  0.0000 fcntl(27, F_DUPFD, 0x00000100)               = 259
17030/1:        97.6627  0.0113  0.0061 pwrite(260, "01 "\0\0\01DA801\001 ,  ".., 1048576, 0x3B500200) = 1048576
17030/1:        97.6654  0.0027  0.0002 pread(259, "01 "\0\0\01DB001\001 ,  ".., 551936, 0x3B600200) = 551936
17030/1:        97.6725  0.0071  0.0023 pwrite(260, "01 "\0\0\01DB001\001 ,  ".., 551936, 0x3B600200) = 551936
17030/1:        97.6734  0.0009  0.0001
stat("/xxx/xqd/dbxs1/log/dbxs1/DBXS1/archivelog/2014_11_19/o1_mf_1_76832_b6rjr5pt_.arc",
0xFFFFFFFF7FFF85B0) = 0
17030/1:        101.0252         3.3518  0.0001
open("/xxx/xqd/dbxs1/log/dbxs1/DBXS1/archivelog/2014_11_19/o1_mf_1_76832_b6rjr5pt_.arc", O_RDWR) = 27
17030/1:        104.3179         3.2927  0.0000 fstatvfs(27, 0xFFFFFFFF7FFF8468)              = 0
17030/1:        104.3181         0.0002  0.0000 lseek(27, 0, SEEK_SET)                       = 0
```

ARC6归档进程

Datagurad Heartbeat进程：其主要作用是不断的ping 远 程 的standby 检测远程以及用来做gap resolution (参见Data Guard Gap Detection and Resolution Possibilities(Doc ID 1537316.1)). 并 且其会不断的相应更新和读取控制文件或者归档文件的信息.

Errors in file /.../.../bdump/dbxs1_arc6_17030.trc:

PING[ARC6]: Heartbeat failed to connect to standby 'rdbxs1'. Error is 1034. ========>>>>>这里可以看出是ARC6 是Heartbeat 进程
PING[ARC6]: Error 3113 when pinging standby rdbxs1.
PING[ARC6]: Heartbeat failed to connect to standby ' rdbxs1 '. Error is 1034.
PING[ARC6]: Heartbeat failed to connect to standby ' rdbxs1 '. Error is 1034.

Control file文件

TYPE	RECORD_SIZE	RECORDS_TOTAL	RECORDS_USED	FIRST_INDEX	LAST_INDEX	LAST_RECID
DATABASE	316	1	1	0	0	0
CKPT PROGRESS	8180	11	0	0	0	0
REDO THREAD	256	8	1	0	0	0
REDO LOG	72	40	25	0	0	78
DATAFILE	428	400	343	0	0	3124
FILENAME	524	2970	365	0	0	0
TABLESPACE	68	400	22	0	0	5
TEMPORARY FILENAME	56	400	23	0	0	30
RMAN CONFIGURATION	1108	50	2	0	0	3
LOG HISTORY	56	4674	4674	2226	2225	77009
OFFLINE RANGE	200	409	0	0	0	0
ARCHIVED LOG	584	149696	147784	145798	143885	167275
BACKUP SET	40	409	84	1	84	84
BACKUP PIECE	736	800	52	1	52	52
BACKUP DATAFILE	116	846	564	1	564	564
BACKUP REDOLOG	76	215	133	1	133	133
DATAFILE COPY	660	818	4	1	4	4
BACKUP CORRUPTION	44	743	0	0	0	0
COPY CORRUPTION	40	409	0	0	0	0
DELETED OBJECT	20	4908	4908	1004	1003	74623

Controlfile有15万的归档信息条目信息，Heartbeat 进程需要读取Controlfile 产生 Controfile sequenetail read 。怀疑是归档条目过多导致。但是其他库也有比这个条目更多的，controlfile文件更大的。

Control file read

◆ 核心库DBXS1（有问题库）的control file read分布：

	EVENT#	EVENT	WAIT_TIME_MILLI	WAIT_COUNT
1	56	control file sequential read	1	8888683
2	56	control file sequential read	2	142814
3	56	control file sequential read	4	466964
4	56	control file sequential read	8	57968
5	56	control file sequential read	16	21551
6	56	control file sequential read	32	24243
7	56	control file sequential read	64	13521
8	56	control file sequential read	128	2713
9	56	control file sequential read	256	624
10	56	control file sequential read	512	1

Control file read

◆ 核心库DBXS2（没有问题库）的control file read分布：

	EVENT#	EVENT	WAIT_TIME_MILLI	WAIT_COUNT
1	56	control file sequential read	1	2475569855
2	56	control file sequential read	2	68727280
3	56	control file sequential read	4	20346309
4	56	control file sequential read	8	1607624
5	56	control file sequential read	16	3129171
6	56	control file sequential read	32	2350273
7	56	control file sequential read	64	413882
8	56	control file sequential read	128	17817
9	56	control file sequential read	256	1392
10	56	control file sequential read	512	469
11	56	control file sequential read	1024	48
12	56	control file sequential read	2048	7
13	56	control file sequential read	4096	2

Arc6 error stack信息抓取：

为了看看具体arc6归档进程调用哪些函数？部署了5s一次的抓arch的error stack，等待下次问题出现时抓取到信息：

```
> more arch_control_file_read.sh
#!/bin/sh
while true
do
sqlplus /nolog @$HOME/tune/arch_control_file_read.sql 2>$HOME/tune/arch_control_file_read_sqlplus.log 1>&2
sleep 5
done

> more arch_control_file_read.sql
connect / as sysdba
set pages 0
set feedback off
set heading off
spool $HOME/tune/a_control_file_read.sql
select /*+ rule */ 'oradebug setospid '||b.spid||chr(10)||'oradebug dump errorstack 3'||chr(10) as run_cmd
from v$session a,v$process b
where a.paddr=b.addr and a.sid in(
select /*+ rule */ blocking_session from v$session ss
where ss.program like '%LGWR%' and ss.event='enq: CF - contention')
and a.PROGRAM like '%ARC%'
and a.EVENT = 'control file sequential read';
spool off

spool $HOME/tune/a_control_file_read.log
@$HOME/tune/a_control_file_read.sql
spool off
exit
```

Arc6 error stack结果：

```
*** 2014-11-24 17:05:29.822   故障时间点
    ksedmp <- ksedmp <- ksdxfdmp <- ksdxcb <- sspuser
        <- sighndlr <- call_user_handler <- sigacthandler <- pread <- pread
         <- skgfqio <- ksfdread1 <- ksfdread <- kccgft_prefetch <- kccrec_rbl
          <- kccext_ugg <- kccrec_read_write <- kccgftcs <- kcc_scan <- kcrrsal
           <- krsfqgap <- kcrrpng <- kcrrwkx <- kcrrwk <- ksbcti
            <- ksbabs <- ksbrdp <- opirip <- opidrv <- sou2o
             <- opimai_real <- main <- start

*** 2014-11-24 19:00:55.716    正常时间点
     Received <- ksedmp <- ksed mp <- ksdxfdmp <- ksdxcb
        <- sspuser <- sighndlr <- call_user_handler <- sigacthandler <- syscall6
         <- sskgpwwait <- ksliwat <- kslwaitns_timed <- kskthbwt <- kslwait
          <- ksarcv <- ksbabs <- ksbrdp <- opirip <- opidrv
           <- sou2o <- opimai_real <- main <- start
```

故障点和正常时候的ARCH Heartbeat 代码路径是不一样，kcc_scan <- kcrrsal <- krsfqgap <- kcrrpng <- 这些函数是和Controlfile 读有关，并且第一个应该是做ping 相关，第二个是检查gap

DTrace收集

1) 这个问题是个瞬时问题，时间很短，只 有几秒，在这瞬间发生的Control file 读问题可能受影响比较多，事务量不 同，应用方式不同，甚至包括IO 性能不同，都会导致进程级在瞬时有变 化. 不同的系统的变化是有个体差异的. 由于是瞬时，这些很细的数据如 IO 方面实际上很难去获取，准备做Dtrace 先行研究一下kcc_scan时间消耗在哪里？

经过多次dtrace抓取，发现并不会每次ping都会产生对kcc_scan的调用。

由于过去了半个月，出于先解决问题处理再找根本原因原则，于是先解决control file文件过大的怀疑点，安排变更将control file重建了一次。

后来经过多次测试，发现kcc_scan只会在Primary和Standby出现gap时，heartbeat才会进行kcc_scan的函数调用。所以这个问题是偶发的；

知道了kcc_scan触发条件后，将灾备停掉20分钟，然后再启动，同时部署dtrace脚本进行收集kcc_scan函数的trace信息。

Dtrace脚本

```
# cat time_func1.d
#pragma D option flowindent
pid$1::$2:entry
{
        self->in = 1;
        self->t = timestamp;
        ustack(50,10);
}

pid$1::kc*:entry,pid$1::ks*:entry
/ self->in /
{
        self->tm[probefunc] = timestamp;
}

pid$1::kc*:return,pid$1::ks*:return
/ self->in /
{
        printf("%d nsec on %s\n", timestamp - self->tm[probefunc], probefunc);
        self->tm[probefunc] = 0;
}

pid$1::$2:return
/ self->in /
{
        self->in = 0;
        printf("%d nsec on this function\n", timestamp - self->t);
        exit(0);
}
```

跟踪arc6进程并且函数是kccscan的堆栈

记录开始执行kccscan函数时间

跟踪kccscan函数中以kc和ks开头的函数，并记录开始时间。

跟踪kccscan函数中以kc和ks开头的函数结束时，打印消耗的时间。

跟踪kccscan函数结束时打印消耗的时间。

Dtrace脚本

```
# cat time_func_stack.d
#pragma D option flowindent
pid$1::$2:entry
{
        self->in = 1;
        self->t = timestamp;
        ustack(50,10);
}

pid$1::kc*:entry,pid$1::ks*:entry,pid$1::pread:entry
/ self->in /
{
        self->tm[probefunc] = timestamp;
        printf("-> entry of %s", probefunc);
}

pid$1::pread:entry
/ self->in /
{
        printf("read %d bytes from %d pos on the file %d", arg2, arg3, arg0);
}

pid$1::kc*:return,pid$1::ks*:return,pid$1::pread:return
/ self->in /
{
        printf("%d nsec on %s", timestamp - self->tm[probefunc], probefunc);
        self->tm[probefunc] = 0;
}

pid$1::$2:return
/ self->in /
{
        self->in = 0;
        printf("%d nsec on this function\n", timestamp - self->t);
        self->t = 0;
        exit(0);
}
```

跟踪函数kccscan的堆栈，并跟踪用户线程的栈。

跟踪kccscan函数中以kc，ks开头的和pread函数，并记录开始时间。

跟踪kccscan函数中以kc，ks开头的和pread函数结束时，打印消耗的时间。

跟踪kccscan函数结束时打印消耗的时间。

Dtrace运行结果

```
# dtrace -s /tmp/time_func1.d 27356 kcc_scan
dtrace: script '/tmp/time_func1.d' matched 12087 probes
CPU FUNCTION
128  -> kcc_scan
          oracle`kcc_scan
          oracle`kcrrsal+0x26c
          oracle`krsfqgap+0xeb0
          oracle`kcrrpng+0x35a4
          oracle`kcrrwkx+0xe64
          oracle`kcrrwk+0x3b0
          oracle`ksbcti+0x3e4
          oracle`ksbabs+0x3a8
          oracle`ksbrdp+0x3f8
          oracle`opirip+0x344
          oracle`opidrv+0x4b0
          oracle`sou2o+0x50
          oracle`opimai_real+0x10c
          oracle`main+0x98
          oracle`_start+0x17c
128  <- ksdpec                      8040 nsec on ksdpec
.......
128  <- ksmpga_update_size              9710 nsec on ksmpga_update_size
......
128  <- kskthbwt                    5700 nsec on kskthbwt
......
128  <- kslwtb_resmgr                 15320 nsec on kslwtb_resmgr
......
128  <- kccgft_prefetch               38488440 nsec on kccgft_prefetch
......
128  <- kccext_ugg                  38560720 nsec on kccext_ugg
......
128  <- kccrec_read_write               38914550 nsec on kccrec_read_write
......
128 | kcc_scan:return               40258180 nsec on kcc_scan
......
128  <- kcc_scan                    40277450 nsec on this function
```

跟踪kccscan函数中以kc，ks开头的函数结束时，打印消耗的时间。

跟踪kccscan函数结束时打印消耗的时间。

Dtrace堆栈运行结果

```
183          | pread:entry        read 16384 bytes from 16384 pos on the file 256
183          <- pread             36210 nsec on pread

183            | pread:entry      read 16384 bytes from 262144 pos on the file 256
183            <- pread           28330 nsec on pread

183            | pread:entry      read 16384 bytes from 294912 pos on the file 256
183            <- pread           28070 nsec on pread

183            | pread:entry      read 16384 bytes from 17235968 （1070000十六进制）pos on the file 256
183            <- pread           48710 nsec on pread

183            | pread:entry      read 622592 bytes from 22691840 （ 15A4000十六进制） pos on the file 256
183            <- pread           25012860 nsec on pread （ 163C000 十六进制）

183            | pread:entry      read 655360 bytes from 22691840 （ 15A4000十六进制） pos on the file 256
183            <- pread           2171790 nsec on pread（ 1644000 十六进制）


183   | kcc_scan:return             29690700 nsec on kcc_scan
183<- kcc_scan                     29710330 nsec on this function
```

期间pread一共读了6次，前4次都一个块16k，速度都很快。后两次读38个块和40个块，速度就降得很厉害，从这里看当读多个块而且pos也比较大时，单次花费的时间总是很长。

Dtrace结果：

1. 前3个pread是在读control file header
2. 后3个都是在kccrec_read_write中调用的，是在读archive log entries
3. 第4个pread是从偏移量17235968（1070000十六进制）读的是归档日志是79728，归档时间是12/4日的。说明arc6进程还是从开始时开始检查的。

```
0106ffc0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0106ffd0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0106ffe0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0106fff0h: 00 00 00 00 00 00 00 00 00 00 00 00 FE DD 15 01 ; ............  ..
01070000h: 15 C2 00 00 00 00 04 1C 31 6B BA 48 FF FF 01 04 ; .?.....1k篪    ..
01070010h: 0E 8D 00 00 00 00 02 B4 00 00 24 8C 33 95 51 BE ; .?....?.$?香?
01070020h: 00 12 00 01 00 01 37 70 B0 BA 3B 99 08 3B 00 00 ; ......7p昂;?;..
01070030h: 31 18 16 25 92 E4 10 A5 08 B7 37 6F 33 95 50 70 ; 1..%掍.??o3昉p
01070040h: 92 EB 1C 05 08 B7 37 70 33 95 51 B1 00 1D 44 BD ; 搠...?p3香?.D?
01070050h: 00 00 02 00 00 01 00 00 CB 5B 17 B9 00 40 00 01 ; ........蔽.?@..
01070060h: 2F 70 61 69 63 2F 78 71 64 2F 6E 65 74 73 6C 69 ; /paic/xqd/netsli
01070070h: 66 65 2F 6C 6F 67 2F 6E 65 74 73 6C 69 66 65 2F ; fe/log/netslife/
01070080h: 4E 45 54 53 4C 49 46 45 2F 61 72 63 68 69 76 65 ; NETSLIFE/archive
01070090h: 6C 6F 67 2F 32 30 31 34 5F 31 32 5F 30 34 2F 6F ; log/2014_12_04/o
010700a0h: 31 5F 6D 66 5F 31 5F 37 39 37 32 38 5F 62 37 7A ; 1_mf_1_79728_b7z
010700b0h: 6F 39 6B 67 32 5F 2E 61 72 63 00 00 00 00 00 00 ; o9kg2_.arc......
010700c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
010700d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
010700e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
010700f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070100h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070110h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070150h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01070160h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
```

Dtrace结果：

1. 第5个pread是从偏移量22691840（ 15A4000十六进制）读的是归档日志是81287，归档时间是12/12日的。

```
015a4130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4150h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4160h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a41a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a41b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a41c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a41d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a41e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a41f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4200h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4210h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4220h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4230h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4240h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4250h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4260h: 00 00 00 00 33 A0 7A 87 00 12 00 01 00 01 3D 87 ; ....3燱?.....=?
015a4270h: B0 BA 3B 99 08 3B 00 00 31 18 16 25 9C 3A F2 DB ; 昂;?;..1..%?蜊
015a4280h: 08 B9 3D 86 33 A0 78 49 9C 3C 45 1B 08 B9 3D 87 ; .??燱I?E..??
015a4290h: 33 A0 7A 7E 00 1D 41 C1 00 00 02 00 00 01 00 00 ; 3燱~..A?.......
015a42a0h: CB 5B 17 B9 00 40 00 01 2F 70 61 69 63 2F 78 71 ; 蔽.?@../paic/xq
015a42b0h: 64 2F 6E 65 74 73 6C 69 66 65 2F 6C 6F 67 2F 6E ; d/netslife/log/n
015a42c0h: 65 74 73 6C 69 66 65 2F 4E 45 54 53 4C 49 46 45 ; etslife/NETSLIFE
015a42d0h: 2F 61 72 63 68 69 76 65 6C 6F 67 2F 32 30 31 34 ; /archivelog/2014
015a42e0h: 5F 31 32 5F 31 32 2F 6F 31 5F 6D 66 5F 31 5F 38 ; _12_12/o1_mf_1_8
015a42f0h: 31 32 38 37 5F 62 38 6F 7A 68 79 6C 32 5F 2E 61 ; 1287_b8ozhyl2_.a
015a4300h: 72 63 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; rc..............
015a4310h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4320h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4330h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
015a4340h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ;
```

Dtrace结果：

1. 第5个pread读了608k个块一直到偏移量22691840（ 1644000 十六进制）读的是归档日志是81474，归档时间是12/14日的。

```
0163fe90h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163fea0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163feb0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163fec0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163fed0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163fee0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163fef0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff00h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff10h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff20h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff30h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff40h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff50h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163ff60h: 00 00 00 00 33 A1 E8 EA 00 12 00 01 00 01 3E 42 ; ....3¤?.....>B
0163ff70h: B0 BA 3B 99 08 3B 00 00 31 18 16 25 C7 54 0C D2 ; 昂;?;..1..%荘.?
0163ff80h: 08 B9 3E 41 33 A1 E7 78 C7 55 3D 56 08 B9 3E 42 ; .?A3 $ x崖=V.?B
0163ff90h: 33 A1 E8 DF 00 1D 41 B1 00 00 02 00 00 01 00 00 ; 3¤?.A?.......
0163ffa0h: CB 5B 17 B9 00 40 00 01 2F 70 61 69 63 2F 78 71 ; 蔽.?@../paic/xq
0163ffb0h: 64 2F 6E 65 74 73 6C 69 66 65 2F 6C 6F 67 2F 6E ; d/netslife/log/n
0163ffc0h: 65 74 73 6C 69 66 65 2F 4E 45 54 53 4C 49 46 45 ; etslife/NETSLIFE
0163ffd0h: 2F 61 72 63 68 69 76 65 6C 6F 67 2F 32 30 31 34 ; /archivelog/2014
0163ffe0h: 5F 31 32 5F 31 34 2F 6F 31 5F 6D 66 5F 31 5F 38 ; _12_14/o1_mf_1_8
0163fff0h: 31 34 37 34 5F 62 38 72 76 33 30 32 23 C4 15 01 ; 1474_b8rv302#?.
01640000h: 15 C2 00 00 00 00 05 90 31 6C 23 BB FF FF 01 04 ; .?....?l#?  ..
01640010h: FC F1 00 00 5F 62 38 72 6F 74 74 6D 6E 5F 2E 61 ;   .._b8rottmn_.a
01640020h: 72 63 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; rc..............
01640030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01640040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
01640050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
```

Dtrace结果：

1. 第6个pread开始的位置和第4个相同，但是读了640k。结尾读的是归档日志是81465，归档时间是12/14日的。

```
0163be90h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bea0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163beb0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bec0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bed0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bee0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bef0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf00h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf10h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf20h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf30h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf40h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf50h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163bf60h: 00 00 00 00 00 00 00 00 00 00 00 00 33 A1 D3 DA ; ............3ϕ?
0163bf70h: 00 12 00 01 00 01 3E 39 B0 BA 3B 99 08 3B 00 00 ; ......>9昂;?;..
0163bf80h: 31 18 16 25 C3 9F 9A 62 08 B9 3E 38 33 A1 D3 D6 ; 1..%脰歇.?83ϕ?
0163bf90h: C3 9F 9A 7F 08 B9 3E 39 33 A1 D3 D7 00 00 01 BD ; 脰?.?93ϕ?..?
0163bfa0h: 00 00 02 00 00 01 00 00 CB 5B 17 B9 00 40 00 01 ; ........蔽.?@..
0163bfb0h: 2F 70 61 69 63 2F 78 71 64 2F 6E 65 74 73 6C 69 ; /paic/xqd/netsli
0163bfc0h: 66 65 2F 6C 6F 67 2F 6E 65 74 73 6C 69 66 65 2F ; fe/log/netslife/
0163bfd0h: 4E 45 54 53 4C 49 46 45 2F 61 72 63 68 69 76 65 ; NETSLIFE/archive
0163bfe0h: 6C 6F 67 2F 32 30 31 34 5F 31 32 5F 31 33 2F 6F ; log/2014_12_13/o
0163bff0h: 31 5F 6D 66 5F 31 5F 38 31 34 36 35 1E 88 15 01 ; 1_mf_1_81465.?.
0163c000h: 15 C2 00 00 00 00 05 8F 31 6C 23 C4 FF FF 01 04 ; .?....?l#?  ..
0163c010h: F8 EE 00 00 5F 62 38 72 6F 74 74 6D 6E 5F 2E 61 ;   .._b8rottmn_.a
0163c020h: 72 63 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; rc..............
0163c030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163c040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
0163c050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
```

Dtrace结果：

1. 虽然dg最多只有20分钟左右的lag，但是arc进程进行gap检查时还是从开始进行检查定位。在定位最近的lag时方法不是很有效。第5次和第6次从相同位置开始检查，并且重复检查两次。
2. 主要时间消耗在38个块的读取pread调用上，用了25ms，占了总时间的92.5%
3. 重建控制文件之前ash采样异常时间读了7次64个块，导致总的读时间变得更长。

这种多块读再加上连续读取还是有风险的。如果在业务高峰期并发量很大的情况下还是可能引发问题。有些库的control file 文件更大，control file read时间更长，但是业务特性不是那种瞬间高并发的，也不会有问题。

同时根据kcc_scan找到两个类似bug 5399901和9788055，bug 5399901相似度很高，同样是ping archive进程，同样是archivelog条目过多。文章中也描述了查找归档日志条目的代码不是很有效，会扫描整个归档记录。但是从dtrace结果看，并不像oracle描述的扫描整改归档记录。

When the number of archived log entries grow to a large value, and dataguard scans these for gap determination, etc, it is possible in some cases for dataguard processes to scan the entire set of archived log entries. The current code that handles the c/f i.o when looking up archived log entries given the record id of the archived log entries is not very efficient, and does not perform read-aheads. This can result in the reader of the archived log entries holding the c/f enqueue in some cases for 2-3 minutes, and this in turn can lead other processes to starve for the c/f enqueue.

案例整改：

知道control file read慢的确切原因后，我们可以制定相关的规范来预防这些问题。对于并发比较高的数据库采用了下列措施：

1. 将online redo file size加大至2GB
2. 将上海DG的日志传输方式由同步方式，改造为级联方式。 即 生产 -> 同城 –> 上海。
3. 针对高并发的操作，使用内存数据库，比如timesten，redis。

总结：

- 更准确
- 瞬时问题
- 高并发
- 操作系统
- IO
- 不是一个直观的调试工具
- 传统数据库和互联网数据库

结语

- 感谢：IT168、ITPUB、DTCC
- 感谢：中国平安DBA团队
- 感谢：在座的每一位伙伴

THANKS

SequeMedia 盛拓传媒 IT168.com ITPUB ChinaUnix