1、Base on the core we can find the process hung when access the inode 0xb000200:

crash> struct nameidata ffff883fb3c6be98 | grep inode inode = 0xffff881f60e87590,crash> crash> struct inode 0xffff881f60e87590 | grep i\_ino i\_ino = 0xb000200,crash> crash> eval 0xb000200hexadecimal: b000200 decimal: 184549888 octal: 1300001000 binary: 0000000000000000000000000000000000001011000000000000001000000000crash>

2、find all extents of this inode:

xfs\_db -f /dev/sds -c "inode 184549888" -c p

……………

……………

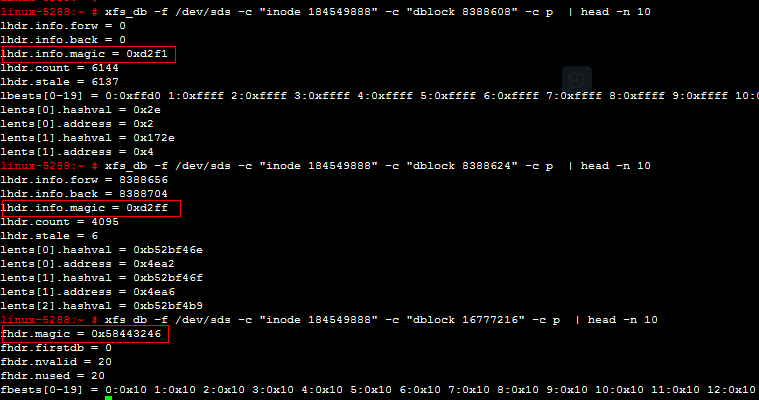
u.bmx[0-27] = [startoff,startblock,blockcount,extentflag] 0:[0,92274928,16,0] 1:[16,92275163,32,0] 2:[48,92274912,16,0] 3:[64,92279954,16,0] 4:[80,92280002,16,0] 5:[96,92295660,16,0] 6:[112,92413160,16,0]

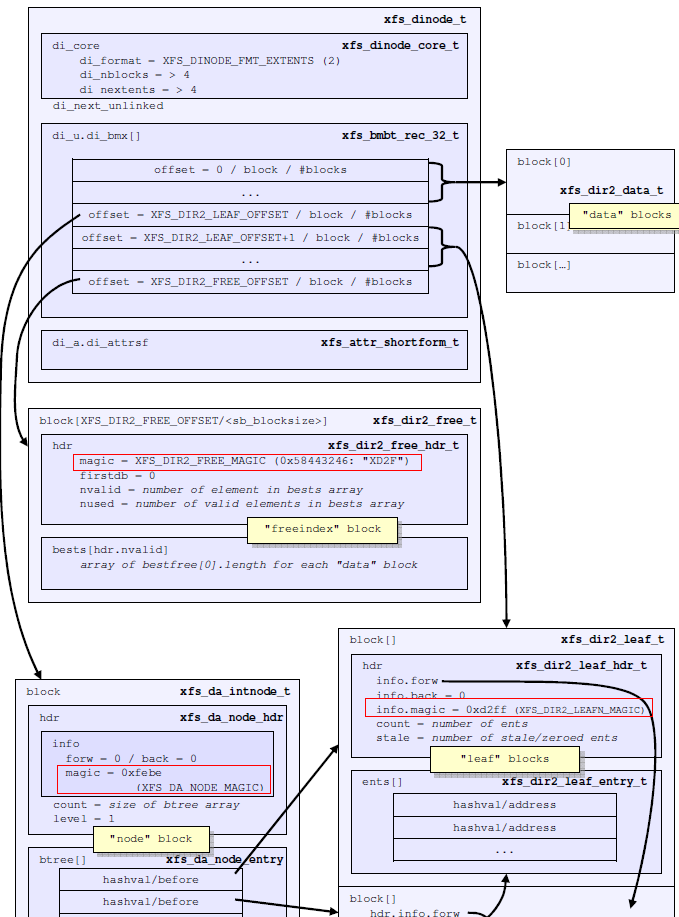
7:[128,92418278,16,0] 8:[144,92515562,16,0] 9:[160,92521441,16,0] 10:[176,92551662,16,0] 11:[192,92557537,16,0] 12:[208,92557664,16,0] 13:[224,92593130,16,0] 14:[240,92610030,16,0] 15:[256,92625378,16,0]

16:[272,92676842,16,0] 17:[288,92718824,16,0] 18:[304,92730849,16,0] 19:[8388608,92274976,16,0] 20:[8388624,92279970,32,0] 21:[8388656,92297967,16,0] 22:[8388672,92467681,16,0] 23:[8388688,92544954,16,0]

24:[8388704,92558059,16,0] 25:[8388720,92655339,16,0] 26:[8388736,92635694,16,0] 27:[16777216,92279938,16,0]

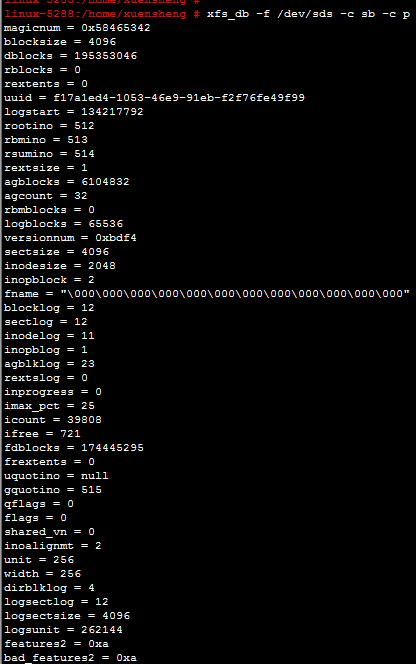
3、The magic of 19:[8388608,92274976,16,0] should be 0xfebe, but now it is 0xd2f1, so we know the meta-data is wrong.



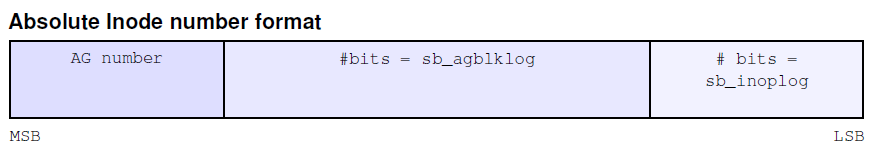


4、 Find out the inode 184549888 is place on No. 0x20057000 sector:

1）collect info of superblock：

****

agblocks=6104832   agblklog = 23  inoplog=1

****

According to the above information of superblock we can know why inode 184549888 is placing in **No.** 0x20057000 sector:

Inode 184549888 is equal to hex 0xB000200

AG number =  0xB000200 >> (23 + 1) = 0xB = 11

ag block = 0x200  >> 1 = 0x100 = 256

AG\_number \* agblocks + ag\_block = 11 \* 6104832   + 256 = 67153408

Block\*(block size/sector size)= 67153408 \* (4096 / 512) = 537227264 = 0x20057000

19:[8388608,92274976,16,0]:

92274976 is equal to hex 0x5800120

AG\_number = 0x5800120 >> 23 = 0xb = 11

ag block = 0x5800120 & (( 1 << 23) – 1) = 0x120

 AG\_number \* agblocks + ag\_block = 11 \* 6104832 + 288 = 67153440

Block\*(block size/sector size) = 67153440 \* （4096 / 512） = 537227520 = 0x20057100

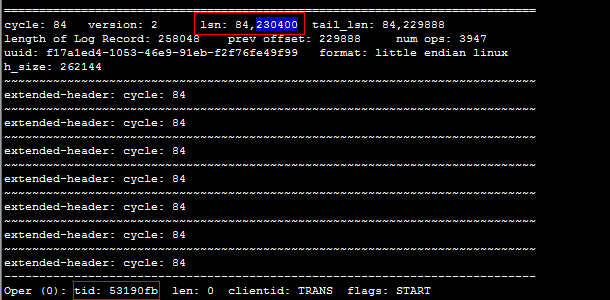
We can find following message in /var/log/message:

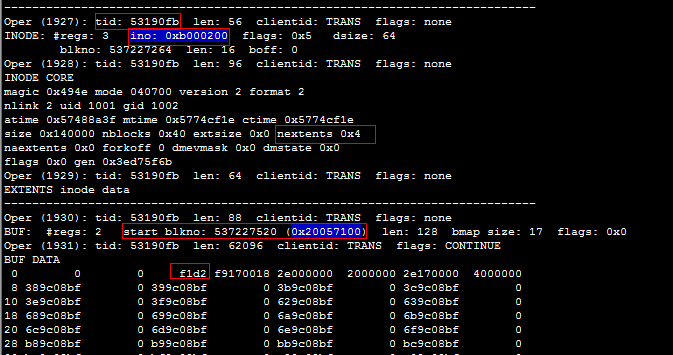
Jun 30 15:50:16 linux-5288 kernel: [1506257.413164] XFS (sds): Device sds: metadata write error block 0x20057000

So we know the inode 184549888 write failed on Jun 30.

5、 Collect information of XFS log about inode 184549888 and block 19:[8388608,92274976,16,0]

Collect the XFS log information via “xfs\_logprint –o /dev/sds”:





According to the above information we can know the nextents of inode 184549888 (0xb00200)should be 0x4, but in fact we find the core.nextents is 28 via following command:

cid:image019.png@01D1DEC0.A823A350

So we know the XFS filesystem recover the inode 184549888 failure.

But why the XFS filesystem recover this inode failed? We continue to analyze this issue:

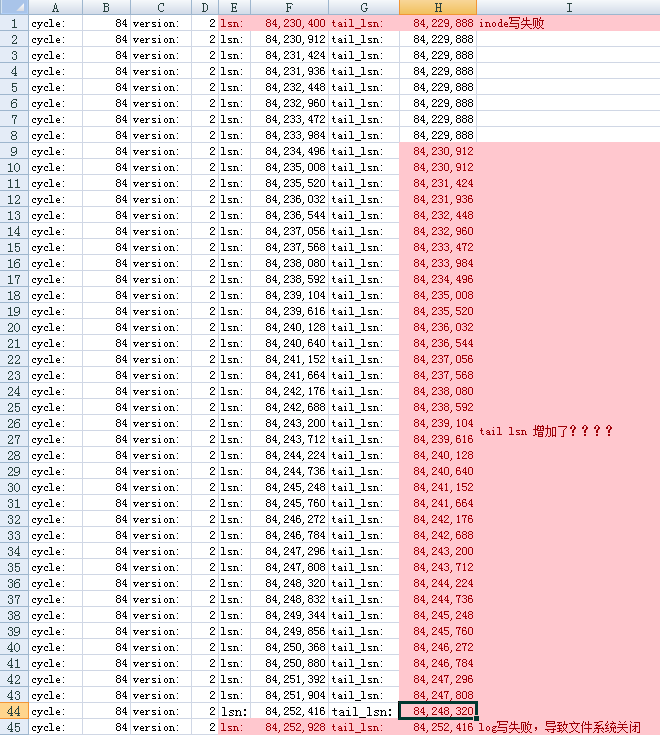
6、

We can find following error messages via /var/log/message:

Jun 30 15:50:27 linux-5288 kernel: [1506268.259835] XFS (sds): I/O error occurred: meta-data dev sds block 0x2e975e00       ("xlog\_iodone") error 5 buf count 262144

Collect the LSN via “xfs\_logprint”:

xfs\_logprint –o /dev/sds | grep lsn

****

cid:image020.png@01D1DEC0.A823A350

134217792 + 252928 / (4096/512) = 134249408 (0x8007bc0)

(0x8007bc0 >>23)=0x10

0x8007bc0 &(( 1 << 23) – 1)=0x7bc0

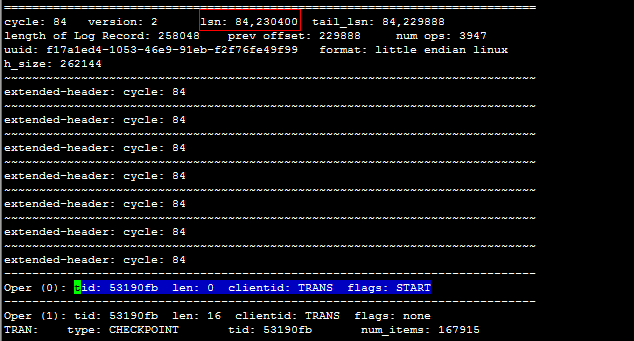
0x10\*6104832(agblocks)+ 0x7bc0=0x5d27000+0x7bc0=0x5d2ebc0

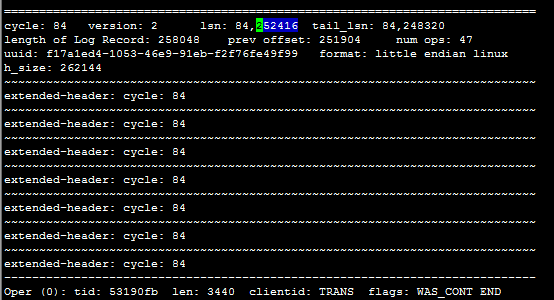
0x5d2ebc0\*（4096 / 512）= 0x2e975e00

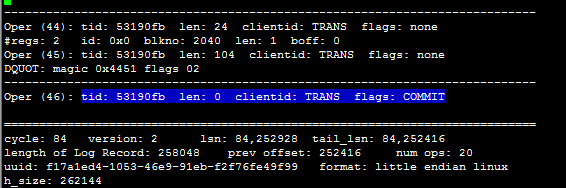
So we know the LSN 84252928 write failed.

7、

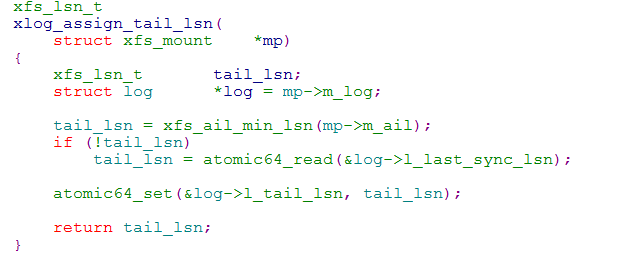
According to the information below, we know the LSN from 84230400 to 84252416 are all on the same transaction



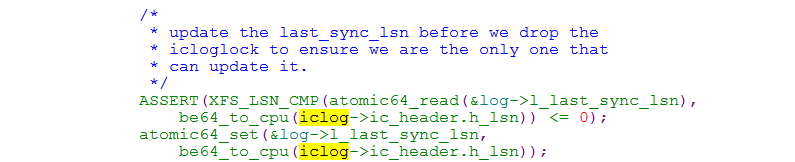




Analyze the “xlog\_assign\_tail\_lsn”, we can know LSN of this LR will become the new tail\_lsn when “xfs\_ail\_min\_lsn” returns 0.

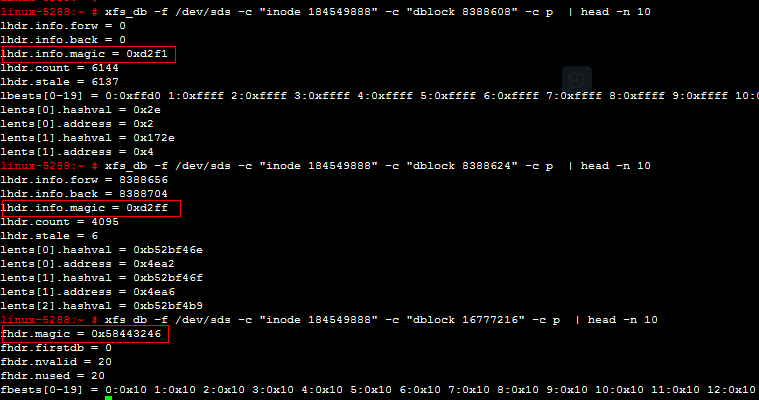


Continue to analyze the “xlog\_state\_do\_callback”, we know the last\_sync\_lsn is updated with iclog’s LSN.

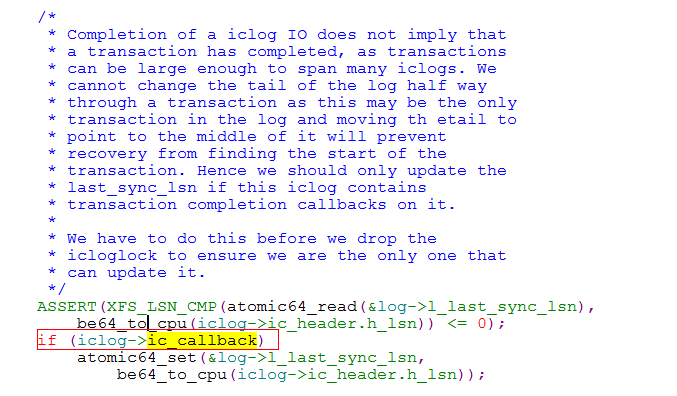


Therefore, when LSN 84252928 write failed, the XFS filesystem only to recover the data from tail\_lsn(84248320) to lsn(84252416), it did not recover the LSN 84230400 although the LSN 84252416 and 84230400 are in the same transaction.

This means the XFS filesystem did not recover the inode 184549888 and block 19:[8388608,92274976,16,0], this lead to the inode 184549888 and dblock 8388608 with an abnormal magic 0xd2f1.



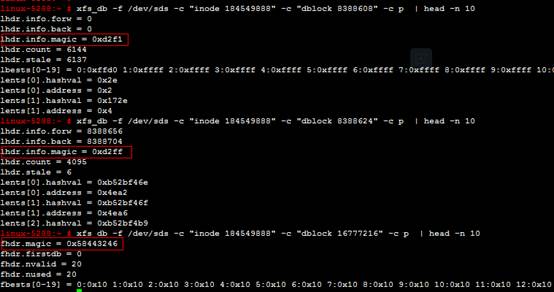
The kernel version 4.6.4 has fixed this issue:



It is related to this bug:

<https://git.geekli.st/pirej/kernel/commit/7e9620f21d8c9e389fd6845487e07d5df898a2e4>

Base on the above analysis we can know the magic of inode 184549888 and dblock 8388608 is 0xd2f1.



But why the process hung when access the "/home/xuensheng/tmpdir/cachefile/0000/" ?  Here is analysis:

1）

crash> bt 13710

PID: 13710  TASK: ffff883fbdfa4400  CPU: 0   COMMAND: "mkmfs:ioc-ctrl"

#0 [ffff883fb3c6b808] schedule at ffffffff8146295b

#1 [ffff883fb3c6b950] schedule\_timeout at ffffffff81463165

#2 [ffff883fb3c6b9e0] \_\_down at ffffffff81463fb8

#3 [ffff883fb3c6ba20] down at ffffffff81089ac7

#4 [ffff883fb3c6ba40] xfs\_buf\_lock at ffffffffa04771e9 [xfs]

#5 [ffff883fb3c6ba60] \_xfs\_buf\_find at ffffffffa047870b [xfs]

#6 [ffff883fb3c6bab0] xfs\_buf\_get at ffffffffa0478856 [xfs]

#7 [ffff883fb3c6baf0] xfs\_buf\_read at ffffffffa0478a14 [xfs]

#8 [ffff883fb3c6bb20] xfs\_trans\_read\_buf at ffffffffa046dd3e [xfs]

#9 [ffff883fb3c6bb60] xfs\_da\_do\_buf at ffffffffa044139f [xfs]

#10 [ffff883fb3c6bc50] xfs\_da\_read\_buf at ffffffffa04418a4 [xfs]

#11 [ffff883fb3c6bc70] xfs\_da\_node\_lookup\_int at ffffffffa04428a1 [xfs]

#12 [ffff883fb3c6bcc0] xfs\_dir2\_node\_lookup at ffffffffa0449cbf [xfs]

#13 [ffff883fb3c6bd00] xfs\_dir\_lookup at ffffffffa0444918 [xfs]

#14 [ffff883fb3c6bdd0] xfs\_lookup at ffffffffa047189f [xfs]

#15 [ffff883fb3c6be20] xfs\_vn\_lookup at ffffffffa047e5a2 [xfs]

#16 [ffff883fb3c6be50] d\_alloc\_and\_lookup at ffffffff81168022

#17 [ffff883fb3c6be70] do\_unlinkat at ffffffff8116ccb9

#18 [ffff883fb3c6bf80] system\_call\_fastpath at ffffffff8146cb92

    RIP: 00007f66a54716b7  RSP: 00007f653bba2ea0  RFLAGS: 00010206

    RAX: 0000000000000057  RBX: ffffffff8146cb92  RCX: 0000000000000000

RDX: 0000000003419e58  RSI: 0000000003419e30  RDI: 00007f653bba3960

2)

  xfs\_da\_node\_lookup\_int

   xfs\_da\_read\_buf

    xfs\_da\_do\_buf

           xfs\_trans\_read\_buf

             xfs\_buf\_read

                   xfs\_buf\_get

                     \_xfs\_buf\_find

                       xfs\_buf\_lock

first, the process 13710 success to obtain the lock “bp->b\_sema” of blkno 8388608 on “xfs\_buf\_lock” and return to “xfs\_da\_node\_lookup\_int”, on “xfs\_da\_node\_lookup\_int” the blkno cannot be updated for the ” blk->magic != XFS\_DA\_NODE\_MAGIC”(blk->magic is 0xd2f1, but XFS\_DA\_NODE\_MAGIC is 0xfebe).

And then, the process 13710 try to request the lock “bp->b\_sema” of blkno 8388608 again on “xfs\_buf\_lock” , this time, it is failure to obtain the lock “bp->b\_sema” of blkno 8388608 and hung. This means that the process is repeated to apply the same lock “bp->b\_sema” of blkno 8388608

#define XFS\_DA\_NODE\_MAGIC      0xfebe

xfs\_da\_node\_lookup\_int(xfs\_da\_state\_t \*state, int \*result)

{

   ………………………..

   ………………………..

if (blk->magic == XFS\_DA\_NODE\_MAGIC) {

                       ………………….

             …………………

                       /\*

                       \* Pick the right block to descend on.

                       \*/

                       if (probe == max) {

                                 blk->index = max-1;

                                 blkno = be32\_to\_cpu(node->btree[max-1].before);

                       } else {

                                 blk->index = probe;

                                 blkno = be32\_to\_cpu(btree->before);

                       }

              }

  …………………..

  …………………..

}

crash>  struct xfs\_da\_state\_blk ffff881f6d283df0

struct xfs\_da\_state\_blk {

  bp = 0x0,

  blkno = 0x800000,

  disk\_blkno = 0x0,

  index = 0x0,

  hashval = 0x0,

  magic = 0x0

}

crash>

crash> eval 0x800000

hexadecimal: 800000  (8MB)

    decimal: 8388608

      octal: 40000000

     binary: 0000000000000000000000000000000000000000100000000000000000000000

crash>

crash> struct inode 0xffff881f60e87590 | grep i\_ino

  i\_ino = 0xb000200,

crash>

crash> eval 0xb000200

hexadecimal: b000200

    decimal: 184549888

      octal: 1300001000

     binary: 0000000000000000000000000000000000001011000000000000001000000000

crash>

crash> struct xfs\_buf\_t ffff881f69f0e680 | grep b\_bn

  b\_bn = 0x20057100,

crash>

crash> eval 0x20057100

hexadecimal: 20057100

    decimal: 537227520

      octal: 4001270400

     binary: 0000000000000000000000000000000000100000000001010111000100000000

crash>