e2fsck heuristics for detecting corrupted inodes

Girish Shilamkar

06 April 2007

1 Requirements

The current e2fsck code does piecemeal fixing of corrupt inodes. This means that if the inode has a bad size, and bad feature flags, and references bad blocks it will happily correct each one in isolation instead of taking a high-level look at the inode and determining it is garbage.

Inorder that e2fsck is able to detect that the complete inode is corrupt, a counter i.e badness counter is maintained which will indicate the extent of corruption of inode. If the badness is above a certain threshold then the inode is deleted.

2 Functional specification

The mechanism for processing bad inodes consists of marking the inode as bad in pass1 [mark_inode_bad()] and then fixing the problem in pass2 [e2fsck_process_bad_inode()]

The new functionality added :

1. The inodes won't just be marked as bad but even their degree of badness will be recorded.
2. More inode fields will contribute towards marking the inode bad, if found bad.

Whenever any field is found to be corrupt in an inode the badness counter will be increased. The cases which indicates corruption in inode are:

```
PR_1_SET_IMAGIC          /* corrupt inode->i_flags */
PR_1_BAD_EA_BLOCK       /* inode->i_alloc corrupt*/
PR_1_INODE_TOOBIG
PR_1_TOOBIG_DIR
PR_1_TOOBIG_REG
```

```
PR_1_TOOBIG_SYMLINK
PR_1_EXTRA_ISIZE      /*bad inode->i_extra_isize */
PR_1_BAD_I_BLOCKS    /* traversed blocks != inode.i_block */ PR_1_BAD_I_SIZE     /* inc
PR_1_ATTR_VALUE_BLOCK  /*Incorrect attr in inode */
PR_1_ATTR_NAME_LEN
PR_1_ATTR_VALUE_SIZE
PR_1_ATTR_VALUE_OFFSET
PR_1_ATTR_VALUE_BLOCK
PR_1_ATTR_HASH
PR_1_SET_EXTENT_FL   /* inode->iflags corrupt */
PR_1_UNSET_EXTENT_FL
PR_1_ILLEGAL_BLK_NUM
PR_1_INDIRECT_BAD
PR_1_EXTENT_BAD
PR_1_EXTENT_IDX_BAD
PR_1_SET_IMMUTABLE
PR_1_SET_NONZSIZE
PR_1_HTREE_NODIR
PR_1_HTREE_SET
PR_1_HTREE_BADROOT
PR_1_COMPR_SET
PR_1_ZERO_LENGTH_DIR
PR_1D_DUP_FILE_LIST  /*Shared block*/
PR_2_BAD_MODE   /*inode->i_mode*/
PR_2_FILE_ACL_ZERO,
```

New checks to be added:

- Check the values of atime, mtime and ctime. The values are checked if set to some futuristic, impossible time. For [am]time check if the values are set before that of ctime. Also check that i_ctime >= sb->s_mkfs_time.
- Check the file size if > 2 TB, as of now a file cannot be bigger than 2TB.
- In pass2 check if inode.i_mode is same as dirent filetype.
- Check if i_blocks ~= i_size/block_size. In case of sparse files the condition will fail but attributing badness of 1 won't delete the inode on its own.

Note: Bad reference count doesn't contribute to badness because this case is found in pass4 and badness is checked in pass2.

3 Use cases

3.1 Sparse file.

- Create a sparse file. Run e2fsck.

- Due to its sparseness the i_size != (i_blocks/block_size) and hence the inode will be marked bad. Check this working
- Ensure that file is not deleted.

3.2 Invalid [amc]time

- Corrupt time fields in the inode with future dates, [am]time set after than that of ctime.
- Check if the these cases were identified by e2fsck.

3.3 Incorrect mode set in i_mode

- Change the i_mode in inode.
- Run e2fsck.
- This condition should be detected in pass2.

3.4 Too many bad blocks.

- Corrupt the i_block, the last triple indirect blocks are corrupted.
- Run e2fsck and insure that inode gets deleted due to too many bad blocks.

3.5 Random corruption.

- Corrupt 0-128 bytes in inode at random location.
- Check if the inode was deleted or not.
- Gather statistics for this and analyse it.

4 Logic specification

The implementation of this feature requires changes to current e2fsck i.e the inode_bad_map bitmsp is re-placed by ext2_icount mechanism. It not only keeps track of bad inodes but also the degree of badness.

In pass1 and pass1b mark_inode_bad() is changed to record the badness, every time one of the cases of inode corruption is found the badness is incremented. The problem is also fixed immediately as till that point of time it is unknown if the inode is corrupted enough to be deleted.

In pass2, if the inode has been marked bad, it is checked if the badness value is above the threshold, if yes the inode is deleted.

How badness is incremented and what is the threshold ?

Badness is incremented every time the following fields in inode are found corrupt.

```
ext2_inode
{
__u16   i_mode;         /* File mode */
__u32   i_size;         /* Size in bytes */
__u32   i_atime;        /* Access time */
__u32   i_ctime;        /* Creation time */
__u32   i_mtime;        /* Modification time */
__u32   i_dtime;        /* Deletion Time */
__u32   i_blocks;       /* Blocks count */
__u32   i_flags;        /* File flags */
__u32   i_block[EXT2_N_BLOCKS]
__u32   i_file_acl;     /* File ACL */
__u32   i_dir_acl;      /* Directory ACL */
}
```

Normally the badness is incremented by 1 for all the fields with following exceptions.

i_mode : If mode is found corrupt the inode is deleted, which is normal course of action.

i_ctime : The badness is incremented by 2 cause, more that [am]time cause , user can change [am]time.

i_flags : For every flag found corrupt badness is incremented by 1.

i_block[] :

- Block Map: e2fsck_ind_block_verify() checks for corruption in i_block[]. If more than 4 block nos are found to be corrupt then the inode is deleted in the current implementation. Badness of 1 for each corrupt block no.
- Extent Map : Every bad extent or extent_idx contributes badness of 2. If the extent header corrupt then the inode is deleted immediately (this is the normal e2fsck behaviour).

The threshold for badness is by default set to 7. It can be over-ridden by passing an extended option to e2fsck.

e.g e2fsck /dev/hda1 -E inode_badness_threshold = 10.

5 State management

5.1 State invariants

The changes which were made to inode fields if they are found to be corrupt are invariants.

5.2 Scalability & performance

If the number of inodes which have bad fields increases too much the current ext2_icount mechanism becomes in-efficient and will also consume more memory. Otherwise it can also improve e2fsck performance by avoiding lengthy duplicate block checking for obviously-corrupt inodes.

5.3 Recovery changes

ext2_icount is an in-memory data structure. Even if it is lost due to crash, the behaviour of e2fsck be same as older one.

5.4 Locking changes

None.

5.5 Disk format changes

None.

5.6 Wire format changes

None.

5.7 Protocol changes

None.

5.8 API changes

mark_inode_bad() changed to e2sck_mark_inode_bad(), which not only marks the inode as bad but also increases the badness of the inode.

e2fsck_process_bad_inode() deletes the inode if the badness is found to be more than 7.

5.9 RPCs order changes

None.