Journal Checksums

Author: Girish Shilamkar

23 May 2007

1 Requirements

Implement journal checksum inorder to ensure the correctness of the journal before it being written down on to the disk. Make changes to e2fsprogs to do checksum based recovery and also ensure that older e2fsprogs or kernels don't recover the journal when they had been checksummed. Make necessary changes to take advantage of checksumming by writing the commit record asynchronously.

2 Functional specification

Add crc32 checksumming of the ext3 journal transaction blocks to the commit block of each transaction. Also allow journal commit block to be submitted at the same time as transaction data blocks (also called *async_commit*).

Add compatibility feature flags to journal superblock (`INCOMPAT_ASYNC_COMMIT` and `COMPAT_CHECKSUM`) to ensure older kernels/e2fsck do not try to recover a journal that had the commit block written before the rest of the transaction.

Add transaction crc32 checksum verification to journal recovery in both the kernel and e2fsck (these both use the same `recovery.c` file so it should be possible to use the same code in both places).

The `INCOMPAT_ASYNC_COMMIT` flag if set, also sets the `COMPAT_CHECKSUM` flag, in order to allow asynchronous commit of commit block. We need to checksum the journal to avoid the recovery mechanism to identify an incomplete transaction as complete one due to the presence of commit block. In a journal with checksum the presence of commit block is no more guarantee of complete transaction.

The ext3 performance, will increase due to asynchronous write of commit block and reliability due to checksum.

3 Use cases

Performance test :

- Compare the time required to write and to recover by setting checksum on/off.
- Compate the time required to write and to recover by setting async_commit on/off.

Sanity Test:

- Normal Recovery.
- Crash the filesystem while writing to disk. Print the transaction numbers along with checksums.
- Recover the filesystem using mount and then with e2fsck.
- Ensure that transaction numbers and checksusms are correct and consistent.
- Recover a fs which has one of the journal blocks corrupt.
- Crash the filesystem while writing to disk.
- Corrupt one of the blocks from any transaction but the last.
- Recover the fs with mount and then with e2sfck.
- Ensure that mount/e2fsck operation is aborted and appropriate error msg is flagged.
- Recover a fs which has corrupt journal block in last transaction.
- Corrupt one of the blocks from last transaction.
- Ensure that mount/e2fsck succeeds with last sane transaction.

4 Logic specification

4.1 Commit:

The function `journal_write_commit_record()` is previously wrote commit block synchronously is split into two parts, `journal_submit_commit_record()` and `journal_wait_on_commit_record(` The call to `journal_submit_commit_record()` is always called, but `journal_wait_on_commit_record()` is not called for asynchronous commits.

```
struct commit_header
{
      __be32 h_magic;
      __be32 h_blocktype;
      __be32 h_sequence;
      unsigned char h_chksum_type;
      unsigned char h_chksum_size;
      unsigned char h_padding[2];
      u32 h_chksum[JBD2_CHECKSUM_BYTES];
};
```

A new header is added to commit block i.e commit_header. The commit_header is actually an extension to the existing header:

```
struct journal_header_s
{
      __be32 h_magic;
      __be32 h_blocktype;
      __be32 h_sequence;
}
```

The checksum is calculated and stored in commit block along with its size and type.

4.2 Recovery:

In PASS_SCAN the blocks described by the descriptor blocks are read and their checksums are calculated. When commit block is found the checksum value stored in the commit header is compared with caluculated checksum. If the checksum verification fails this means that either one or more blocks in the transaction are corrupt or it is a interrupted commit. In order to know the difference we check if the next transaction contains commit block, if yes then it was interrupted commit else corruption. In case of corruption an error message is flagged about the possible corruption.

An error is marked if corruption is detected, this behaviour might change after consultation with other ext4 developers.

5 State management

5.1 State invariants

5.2 Scalability & performance

Use of checksum allows to write the commit block asynchronously. This should speed up the writing to journal. During writing and recovery, checksums are calculated which will require more processing than usual, effect of this is to be verified in unit testing.

During recovery the blocks are not immediately written to disk after read from the journal. instead written out only after verification. This means all transaction blocks are read into memory and then collectively written back, as oppose to previous behaviour where each block was read from journal and submitted to disk immediately. But since transaction may span more than few MBs this shouldn't require lot of memory.

5.3 Recovery changes

In PASS_REPLAY writing of blocks is delayed till the checkum is verified.

5.4 Locking changes

NA.

5.5 Disk format changes

Commit header extended as mentioned above.

5.6 API changes

journal_write_commit_record() which use to write commit block synchronously is split into two parts i.e. journal_submit_commit_record() and journal_wait_on_commit_record() in order to write it asynchronously.