Multiple Mount Protection

Kalpak Shah

2007-04-03

1 Introduction

The multiple mount protection (MMP) feature is intended to protect the filesystem from being mounted more than once simultaneously. It will also protect changes by e2fsprogs to the filesystem if the filesystem is mounted. This assumes high importance in a shared storage environment (for example, when a OST and a failover OST share a partition).

2 Requirements

The ext3 superblock will get a new field s_mmp_block which will save the heartbeat information. This heartbeat block will be written every N seconds by ext3 with the help of a newly created kernel thread "kmmpd". This block will be sequentially updated starting with 1 after every successful mount. Before actual use, ext3 and e2fsprogs can scan the heartbeat block to make sure that nobody else is currently using the filesystem. If the filesystem is cleanly unmounted/closed then ext3 or e2fsck will write a special magic number (to indicate clean unmount) into the block to speed up mount/open later.

Old kernels should still be able to mount filesystems which use MMP by clearing the FEATURE_INCOMPAT_MMP flag by using tune2fs if they were cleanly unmounted.

3 Functional specification

ext3 superblock will get 2 new fields:

1) s_mmp_block - save the block number which is acting as the heartbeat block

2) s_mmp_interval - interval to update the heartbeat block

#define EXT3_MMP_MAGIC 0x004D4D50 /* ASCII of MMP */

#define EXT3_MMP_SEQ_CLEAN 0xFF4D4D50 /* value of mmp_seq for clean

* unmount */

#define EXT3_MMP_SEQ_FSCK 0xE24D4D50 /* value of mmp_seq when being

* fscked */

#define EXT3_MMP_SEQ_MAX 0xE24D4D4F /* maximum valid sequence number */

struct mmp_struct {

__le32 mmp_magic;

__le32 mmp_seq;

__le64 mmp_time;

char mmp_nodename[64];

char mmp_bdevname[32];

__le16 mmp_interval;

__le16 mmp_pad1;

__le32 mmp_pad2;

};

/* Default interval in number of seconds to update the MMP sequence number. */

#define EXT3_MMP_DEF_INTERVAL 5

* The corruption in the heartbeat block can be found with the help of the magic number. Also if the structure changes then a MMP_MAGIC_V2 can be used to identify the MMP structure version.

* The mmp_seq is a 32 bit integer which is incremented every 5 seconds or so. If a filesystem is cleanly unmounted it should be set to MMP_SEQ_CLEAN. Whenever a filesystem gets successfully mounted then mmp_seq can be started from 1 and this will be perfectly safe.

* mmp_time stores the time when the heartbeat block was last updated. It can be used to tell the user information about when the filesystem was last touched by a node.

* mmp_nodename is the hostname of the running kernel (as found in init_uts_ns.name.nodename) for reporting to the user. With the help of this name the user can understand which node is racing for mounting the filesystem.

* mmp_bdevname gives the block device on which the filesystem which last updated the heartbeat block is mounted.

The updation of the heartbeat block is done by creating a kernel thread after the filesystem is successfully mounted. We add a “struct task_struct *s_mmp_task” to the sbi and create the kmmpd (kernel MMP deamon) by:

sbi->s_mmp_task = kthread_run(kmmpd, mmp_bh, “kmmpd”);

The design for kmmpd will be as follows:

```
static int kmmpd(void *data)
{
struct buffer_head *bh = (struct buffer_head *) data;
struct mmp_s *mmp = (struct mmp_s *) bh->b_data;
u32 seq ;
while (!kthread_should_stop()) {
if (++seq >= EXT3_MMP_SEQ_MAX)
seq = 1;
mmp->mmp_seq = cpu_to_le32(seq);
mmp->mmp_time = cpu_to_le64(get_seconds());
write the mmp block;
last_updated_time = get_seconds();
schedule_timeout_interruptible(mmp_interval secs);
if (get_seconds - last_updated_time > 2 * mmp_interval) {
re-read heartbeat block;
if (old_seq != current seq)
ext3_error("Another node managed to mount the
filesystem");
}
}
/* kthread_stop has been called. Exiting kmmpd..... */
mmp->mmp_seq = EXT3_MMP_SEQ_CLEAN;
mmp->mmp_time = cpu_to_le64(get_seconds());
write the mmp block;
return 0;
}
```

If a node is stuck for more than 2 * mmp_interval, the kmmpd should read the heartbeat block from disk and make sure that mmp_seq has not changed. If it has changed it means that a different node has somehow been able to mount the filesystem. If mmp_seq has changed then ext3_error() should be called by which ext3 will try its best to make the filesystem read-only.

A “-o mmp” option will be added to tunefs to set/unset the MMP feature.

tunefs should also be able to set a mmp_interval by using a “-p <num_secs>” option. With this option, users can set any mmp_interval they deem fit.

Scripts should be able to find the current on-disk data in the MMP heartbeat file. This can be done by adding a feature to debugfs to dump the MMP file contents. Something like “debugfs -R dump_mmp” which will output the details in an easily-parsable form.

While submitting the heartbeat block for writing we will use WRITE_SYNC flag to speed up the on-disk write of the block. But the WRITE_SYNC flag unplugs the device queue and may affect performance if done every mmp_interval seconds. This can be checked during performance testing.

4 Use cases

1. Two mounts at same time
2. One mount delayed by write interval
3. One mount delayed by 2x mount interval
4. One mount delayed by > 2x mount interval
5. Mount during unmount of first filesystem
6. Mount after clean unmount
7. Mount after reboot (can be simulated with dev_read_only facility or just reading s_mmp_block from superblock and doing "dd" to that block before mount)
8. Mount during e2fsck (should never succeed)
9. Mount after aborted e2fsck (should never succeed)
10. e2fsck with mounted fs
11. kmmpd should detect if node is stuck for more than 2 * mmp_interval seconds. After re-reading the heartbeat block, if mmp_seq has changed then kmmpd should call ext3_error()

5 Logic specification

The ext3_multi_mount_protect() function will have the following logic:

1) Read the mmp block

2) Check if mmp block is corrupt, if corrupt then exit with failure

3) Check mmp_seq == EXT3_MMP_SEQ_CLEAN? yes, goto 7

4) Wait 2 * mmp_interval secs

5) Reread the mmp block

6) mmp_seq changed? yes = exit

7) Write a new random sequence number

8) Wait 2 * mmp_interval secs

9) Reread mmp block (make sure to dirty the block else a diskread won't occur)

10) mmp_seq changed? yes = exit

11) start kmmpd, mount successful.

ext3_multi_mount_protect() will be called from ext3_fill_super() if INCOMPAT_MMP feature is set. If this feature is set then older kernels cannot mount this filesystem. They would have to clear this feature by using "tune2fs -O ^mmp" and then they would be able to mount on older kernels also.

6 State management

6.1 State invariants

1) If node is stuck for more than 2 * mmp_interval, the kmmpd should re-read the heartbeat block and make sure that the mmp_seq has not changed.

2) If e2fsck crashes then the filesystem cannot be mounted since the kernel will feel that fsck is being run on the filesystem. In that case, tune2fs - O ^mmp followed by tune2fs -O mmp should be used to disable and then enable the feature to clear the heartbeat block.

3) In case tune2fs is run with -f option then MMP check is skipped.

6.2 Scalability & performance

Having to write the heartbeat block every N seconds may affect performance. The performance impact of the heartbeat block update should be assessed for IO and meta-data using tools like IOR, mdsrate. Also we are using WRITE_SYNC flag which can potentially affect the performance.

6.3 Recovery changes

If the MMP feature is set it will delay remounting by atleast 2 * mmp_interval seconds. By default this will mean a delay of 10 seconds.

6.4 Locking changes

The buffer containing the MMP file should be properly locked when being read/written. Other than this no locking considerations arise here.

6.5 Disk format changes

A EXT3_FEATURE_INCOMPAT_MMP feature has been added to ext3.

The s_mmp_interval and s_mmp_block fields will be added to the ext3 superblock. This has already been accepted upstream.

The MMP structure that will be written into the s_mmp_block has been mentioned above.

6.6 Wire format changes

None.

6.7 Protocol changes

None.

6.8 API changes

None.

7 Alternatives

None.

8 Focus for inspections

None.