Silicon Graphics, Inc.

XFS Overview & Internals
11 - Repair

November 2006

Repairing Filesystems

- Filesystems can be corrupted by
  - Hardware errors
    - Media errors are common
    - Disks are getting bigger and bigger
  - To a muhc lesser degree, bugs in the filesystem

- Filesystems are able to “repair” themselves since they consist of lists, links and reference counts that can be validated
  - But not all information is always recovered, inodes that do not have a parent directory is common due to the directory structure being corrupted

xfs_check

- xfs_check is a script that runs xfs_db to do a filesystem check.
- The "check" command in xfs_db scans all the metadata structures for inconsistency
- xfs_check uses a different codebase to xfs_repair
  - xfs_check and xfs_repair can be used to cross check each other
    - xfs_check vs xfs_repair -n)

xfs_repair

- xfs_repair scans the filesystem and corrects any problems encountered.
- xfs_repair performs a scan and repair in seven phases.
- Each phase relies on the previous phase to fix a certain class of potential errors.
- xfs_repair uses libxfs which is a partial port of the XFS kernel code to user-space.

xfs_repair – Phase 1

- Find, verify and fix superblocks.
- If a superblock is not found, xfs_repair will stop.
- Sets up a virtual mount structure for the common XFS code base (libxfs) to work from.

xfs_repair – Phase 2

- Checks the AG header structures (AGI, AGF and AGFL) and scans the AGF and AGI btrees.

xfs_repair – Phase 3

- Using the AGI btree from phase2, scan the inode tree, processing the unlinked list for deleted inodes and finding possible missing inode clusters.
- Walk all the found inodes, recording used filesytem blocks (extents).
- For directory inodes, scan the directory structure for more lost inodes.
- Any bad inodes are trashed including unrecoverable corrupted directories.

xfs_repair – Phase 4

- Scan inode extents again. Any inode with an extent covering used data is trashed.

xfs_repair – Phase 5

- Rebuild AG headers and structures including the AGI btree, AGF btrees and AGFL regardless whether any errors have been found or not.
- Realtime inodes are also reconstructed.

xfs_repair – Phase 6

- At this stage, the filesytem is in a mountable state.
- Scan the directories analysing all data.
  - Any directories with any corruption are rebuilt with whatever entries can be recovered.
  - A missing root directory is recreated.
  - All inodes that are in a directory are marked reached.
- At the end, any unreached inodes are put into lost+found.

xfs_repair – Phase 7

- nlinks for inodes are corrected based on the data collected in phase 6.

Triaging xfs_check and xfs_repair problems

- Most of the time, inode information is required:

```
> inode <inode number>
> print
```

- The root inode number can be derived from the superblock:

```
> sb 0
> print rootino
```

- For directories, we can also dump the contents from the extent list shown in the inode:

```
> dblock <file offset in blocks>
> print
```

- Directories have file offsets typically starting at 0, 8388608 and 16777216. Each of these offsets stores different information for a directory.
- The filename and inode numbers at 0, hash values at 8388608 and free space information at 16777216.

xfs_repair and xfs_check should agree

- If one of the tools reports a problem when the other passed the filesystem, there is a problem with one of the tools
  - most likely xfs_repair
- http://oss.sgi.com/bugzilla/show_bug.cgi?id=723
- xfs_check finds some errors on the filesystem:

```
link count mismatch for inode
  387655 (name ?), nlink 0,
  counted 2
link count mismatch for inode
  13313696 (name ?), nlink 0,
  counted 2
link count mismatch for inode
  17197100 (name ?), nlink 0,
  counted 2
```

- xfs_repair reports no problems:

```
Phase 1 - find and verify superblock...
Phase 2 - using internal log
        - zero log...
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan and clear agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
        - agno = 4
- process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - clear lost+found (if it exists) ...
        - clearing existing "lost+found" inode
        - marking entry "lost+found" to be deleted
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        - agno = 2
        - agno = 3
        - agno = 4
        - Phase 5 - rebuild AG headers and trees...
        - reset superblock...
Phase 6 - check inode connectivity...
        - resetting contents of realtime bitmap and summary
   inodes
        - ensuring existence of lost+found directory
        - traversing filesystem starting at / ...
rebuilding directory inode 128
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify and correct link counts...
done
```

Dump the offending inodes…

```
# xfs_db -c "inode 387655" -c "print" /dev/sda6
core.magic = 0x494e
core.mode = 040755
core.version = 1
core.format = 1 (local)
core.nlinkv1 = 0
...
core.size = 6
core.nblocks = 0
core.extsize = 0
core.nextents = 0
...
next_unlinked = null
u.sfdir2.hdr.count = 0
u.sfdir2.hdr.i8count = 0
u.sfdir2.hdr.parent.i4 = 135
```

Mount and Repair Fails – Corrupted Log

- If the log is corrupted you will see an error like:

```
# mount <filesystem>
mount: Unknown error 990
# dmesg | tail -20
Filesystem "<filesystem>": xfs_inode_recover: Bad inode magic number . . .
Filesystem "dm-0": XFS internal error xlog_recover_do_inode_trans(1) at line 2352 of file fs/xfs/xfs_log_recover.c.
    Caller 0xffffffff88307729
XFS: log mount/recovery failed: error 990
XFS: log mount failed

# xfs_repair <device>
  Phase 1 - find and verify superblock...
      Phase 2 - using internal log
                - zero log...
      ERROR: The filesystem has valuable metadata changes in a log which needs to
      be replayed.  Mount the filesystem to replay the log, and unmount it before
      re-running xfs_repair.  If you are unable to mount the filesystem, then use
      the -L option to destroy the log and attempt a repair.
      Note that destroying the log may cause corruption -- please attempt a mount
      of the filesystem before doing this.
```

- Useful information can be collected for triage:

```
# /usr/sbin/xfs_logprint –C <filename> <device>
# /usr/sbin/xfs_logprint -t <device>
```

- But in this case, the only option may be to throw the log away:

```
# xfs_repair –L
```