SILICON GRAPHICS | The Source of Innovation and Discovery™

Silicon Graphics, Inc.

XFS Overview & Internals 06 - Allocators

November 2006

Allocation Policy

- The default allocation behaviour of XFS for new files is to place them in the same allocation group as their parent directory.
- Since files and their parent directories are often accessed in close succession, this minimises costly disk seeks.
- The allocator will also attempt to place newly created directories in different allocation groups.
- Combined, these policies help group directories of files together on disk, even if they're being written to concurrently.
- Allocation policies in XFS can change due to
  - 32 bit inode numbers are used on large file systems
  - Some mount options

Allocation Policy - Directories

- New directories are placed in different AGs where possible
- Watch the inode numbers as directory inodes are created:

```
> mkdir a b
> ls -li
total 0
     131 drwxr-xr-x 2 sjv users 6 2006-10-20 12:12 a
33554624 drwxr-xr-x 2 sjv users 6 2006-10-20 12:12 b
```

Allocation Policy - Files

- Files are created in the same AG as their parent directory where possible, which is also evident in their inode numbers:

```
> touch a/1 b/1 a/2 b/2
> ls -1id * */*
     131 a
     132 a/1
     133 a/2
33554624 b
33554625 b/1
33554626 b/2
```

Inode Numbers

- Every inode on disk has a unique inode number associated with it.
- It is a requirement that inode numbers be persistent across unmounts and reboots, so once an inode is written to disk its inode number is fixed.
- For performance reasons it must be possible to quickly find an inode on disk using its inode number.
- XFS uses the physical location of the inode on disk to encode the inode number, which makes finding the inode on disk using the inode number a trivial task.

Inode Number Format

- An inode's location consists of three distinct parts
  - an allocation group number
  - a file system block number within its AG
  - an inode number inside that file system block
- The number of bits required to store each of these values varies with the filesystem geometry
- Larger filesystems can easily require more than 32 bits, which can limit inode allocation to a region at the start of the volume

Inode Number Size

- File systems aren't free to use inode numbers of arbitrary size.
- Operating system interfaces and legacy software products often mandate the use of 32 bit inode numbers even on systems that support 64 bit inode numbers.
- This can be a problem on large file systems, since 32 bit inode numbers only provide enough bits to encode inode locations in the first 1TB of a volume when 256 byte inodes are used, up to 8TB in the case of 2kB inodes.
- For best performance, a file system needs to keep a file's data blocks close to its inode to minimise seeks when performing I/O. XFS's ability to do this suffers on large volumes when 32 bit inode numbers are used.

32bit and 64bit Inodes

- By default, XFS will use 32 bit inode numbers.
- If the system supports it, the -o inode64 option to mount to allow 64 bit inode numbers.
- Once an inode has been written somewhere on the disk that requires a 64 bit inode number, the file system can no longer be used with 32 bit inode numbers
  - The inode64 mount option should not be removed once used
- (IRIX can move inodes to 32 bit numbers with `xfs_reno`, this tool has not been ported to Linux, yet)

32bit and 64bit Inodes

- Inode numbers are stored in big endian format on disk, and host endian format in-core.
- Applications that pass 64 bit inode numbers using 32 bit variables will truncate the 32 most-significant bits.
- Since XFS stores the AG number an inode belongs to in the most significant bits, a result of this truncation can be an inode number that points to an inode in a lower AG by mistake.
- Using that inode number will result in either a lookup on the incorrect inode, or the referencing of an area on disk that doesn't contain inodes at all.

32 bit Inodes on >1TB Filesystems

- When 32 bit inode numbers are used on a volume larger than 1TB in size, several changes occur.
- A 100TB volume using 256 byte inodes mounted in the default inode32 mode has just one percent of its space available for allocating inodes.
- XFS will reserve the first 1TB of disk space exclusively for inodes to ensure that the imbalance is no worse than this due to file data allocations.
- It is no longer possible for file data to reside in the same AG as the parent directory's inode.
- XFS will instead "rotor" through the upper AGs as it allocates space for files, putting each file in a new AG to evenly spread the I/O load.

Rotor Step

- The performance of some workloads will suffer from the distribution each file in a different AG, so the "rotor step" sysctl was added adjust this behavior
- For example, to keep at least a second of ingested 24fps video files in the same AG before moving to the next AG:

```
# sysctl fs.xfs.rotorstep
fs.xfs.rotorstep = 1
# sudo sysctl -w fs.xfs.rotorstep=24
fs.xfs.rotorstep = 24
```

- Note that the rotorstep value is a global one, so setting it will affect the behaviour of all mounted file systems over 1TB in size that use 32 bit inode numbers.

Realtime Allocator

- Certain classes of applications require deterministic latencies on file allocation operations
  - The performance of the standard XFS allocator varies depending on the internal data structures used to manage the filesystem content
- The realtime allocator uses a bitmap algorithm that gives consistent allocation latencies regardless of the filesystem's contents.
- By using the realtime allocator in conjunction with an external log volume, it's possible to remove most of the unpredictability in disk response times that's caused by metadata overheads.

Realtime Allocator Limitations

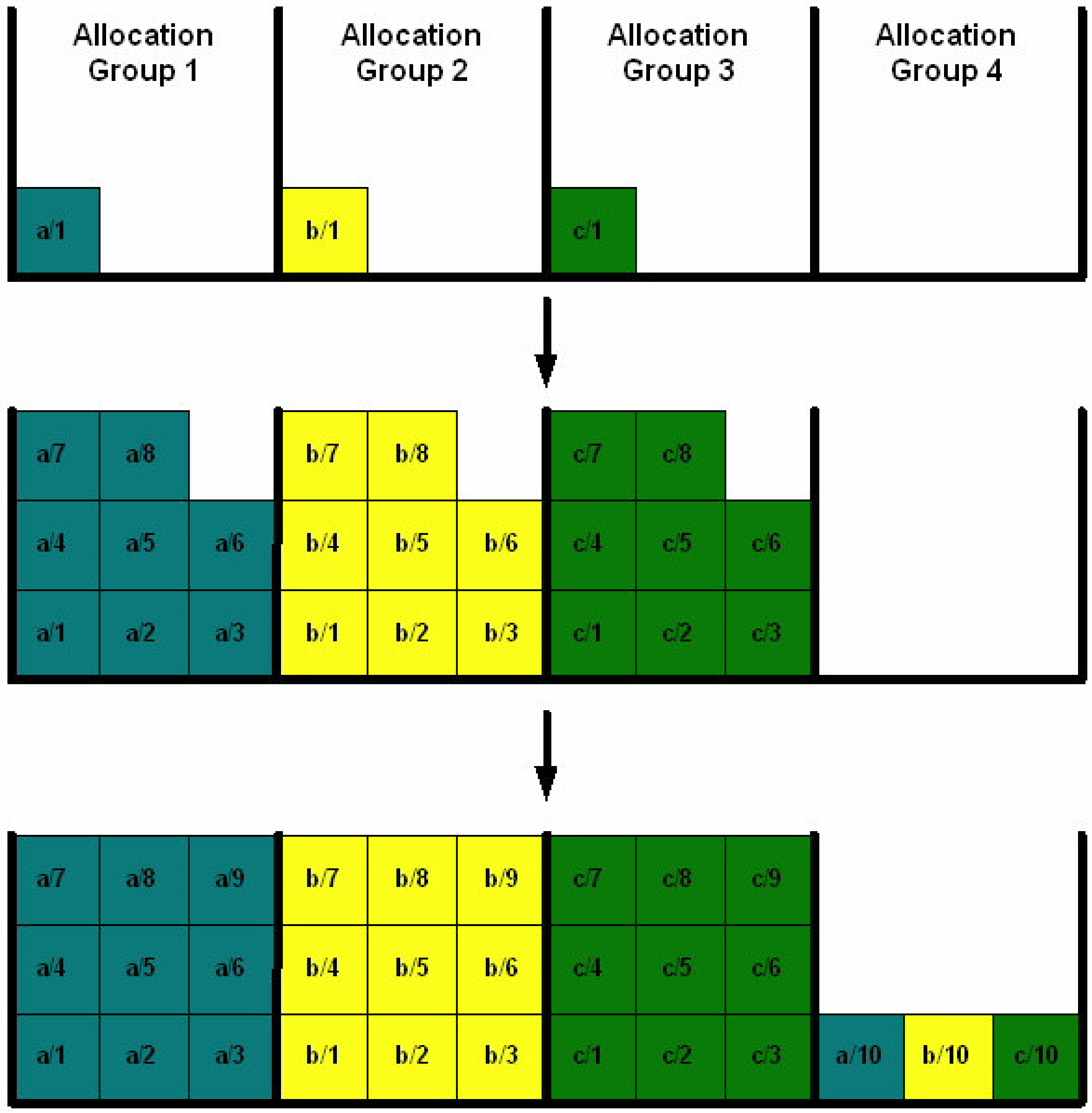
- In practice, the realtime allocator is not widely used.
- It effectively uses a single large allocation group with a single set of data structures losing the parallelism of XFS's allocation groups
  - The locks associated with this central data structure result in the serialisation of concurrent operations to a realtime device
- The realtime allocator is incapable of maintaining a spatial separation on disk for concurrent operations
  - It tries to start new files at random points in the bitmap to reduce this problem but this has a negative impact on some workloads

Traditional allocator impact on some workloads

- A certain class of applications will write many large files to a directory in sequence.
- A film scanner ingesting video may write each video frame to a separate file. To playback the video frames in realtime its important that these files are contiguous on disk for optimal read-ahead performance by the hardware RAID.
- At 24 frames per second, each frame is needed every 40ms, so it is important to keep the disks busy and reading into cache the next frame to be displayed

Traditional Stream Allocation

- Initially each directory is allocated to a separate AG
- Each stream writes to that AG until it is full
- Additional allocations now go in the next consecutive AG that has enough free space
- Multiple streams will start writing to the same AG, interleaving their files and negating any read-ahead


Allocation Group 1
Allocation Group 2
Allocation Group 3
Allocation Group 4
a/1
b/1
c/1
a/7 a/8 b/7 b/8 c/7 c/8
a/4 a/5 a/6 b/4 b/5 b/6 c/4 c/5 c/6
a/1 a/2 a/3 b/1 b/2 b/3 c/1 c/2 c/3
a/7 a/8 a/9 b/7 b/8 b/9 c/7 c/8 c/9
a/4 a/5 a/6 b/4 b/5 b/6 c/4 c/5 c/6
a/1 a/2 a/3 b/1 b/2 b/3 c/1 c/2 c/3 a/10 b/10 c/10

RAID performance with interleaved streams

```
System Performance Statistics
               All Ports     Port 1     Port 2     Port 3     Port 4
 Read  MB/s:      183.9       45.8       46.8       46.3       45.0
 Write MB/s:        0.0        0.0        0.0        0.0        0.0
 Total MB/s:      183.9       45.8       46.8       46.3       45.0

 Read  IO/s:        520        133        129        129        129
 Write IO/s:          0          0          0          0          0
 Total IO/s:        522        134        131        128        129

 Read Hits:         1.3%       1.6%       2.2%       0.6%       0.6%
 Prefetch Hits:     0.8%       1.1%       1.1%       0.6%       0.6%
 Prefetches:       46.3%      46.3%      46.0%      46.1%      46.8%
 Writebacks:        0.0%       0.0%       0.0%       0.0%       0.0%
 Rebuild MB/s:      0.0        0.0        0.0        0.0        0.0
 Verify MB/s:       0.0        0.0        0.0        0.0        0.0

                   Total      Reads     Writes      Pieces   Reads     Writes
 Disk IO/s:          518        518          0        1:      4910          0
 Disk MB/s:        544.6      544.6        0.0        2:     29890          0
 Disk Pieces:      65710      65710          0        3:       340          0
 BDB Pieces:                      0                   4:         0          0
                                                      5:         0          0
  Cache Writeback Data:     0.0%                      6:         0          0
  Rebuild/Verify Data:      0.0%    0.0%              7:         0          0
  Cache Data locked:        0.0%                      8:         0          0
```

With only 1.3% read cache hits, RAID is reading **545MB/s** to return 184MB/s to the client (200% backend overhead)

Filestreams Allocator

- A new allocation algorithm was added to XFS that associates a parent directory with an AG until a preset inactivity timeout elapses.
  - Not in SLES10, planned for SLES10SP1

- A stream that moves to a new AG will cause that AG to be locked, so other streams looking for a new AG will not use the same AG

- The new algorithm is called the Filestreams allocator and it is enabled in one of two ways:
  - the filesystem is mounted with the -o filestreams option, or
  - the filestreams chattr flag is applied to a directory to indicate that all allocations beneath that point in the directory hierarchy should use the filestreams allocator

- Filestreams will have a negative impact on workloads that continue to grow files in the same directory, causing more fragmentation than the default allocator

RAID performance with filestreams

```
System Performance Statistics
                All Ports     Port 1      Port 2      Port 3      Port 4
 Read  MB/s:       299.1        74.0        74.7        75.1        75.2
 Write MB/s:         0.0         0.0         0.0         0.0         0.0
 Total MB/s:       299.1        74.0        74.7        75.1        75.2

 Read  IO/s:         840         209         210         211         210
 Write IO/s:           0           0           0           0           0
 Total IO/s:         836         209         210         208         209

 Read Hits:        99.5%       98.3%       99.6%      100.0%      100.0%
 Prefetch Hits:    98.8%       97.6%       98.9%       99.6%       99.0%
 Prefetches:       42.0%       41.5%       42.0%       42.9%       41.7%
 Writebacks:        0.0%        0.0%        0.0%        0.0%        0.0%
 Rebuild MB/s:      0.0         0.0         0.0         0.0         0.0
 Verify MB/s:       0.0         0.0         0.0         0.0         0.0

                   Total       Reads      Writes      Pieces   Reads     Writes
 Disk IO/s:          614         614           0          1:   39068          0
 Disk MB/s:        345.5       345.5         0.0          2:     111          0
 Disk Pieces:      39290       39290           0          3:       0          0
 BDB Pieces:                       0                      4:       0          0
                                                          5:       0          0
  Cache Writeback Data:     0.0%                          6:       0          0
  Rebuild/Verify Data:      0.0%    0.0%                  7:       0          0
  Cache Data locked:        0.0%                          8:       0          0
```

Almost all data now found in RAID cache, only 15% backend disk I/O overhead

Fragmentation

- Despite the use of extents and the various allocation schemes, XFS files and filesystems may still become fragmented over time
- xfs_db can display the level of fragmentation in the filesystem
  - xfs_db -r /dev/sda3
    - frag -f   file fragmentation percentage
    - frag -d   directory fragmentation percentage
    - freesp   freespace

Fragmentation Example

```
> xfs_db -r device
xfs_db: freesp
   from      to extents  blocks    pct
      1       1   94807   94807   1.36
      2       3   63041  145012   2.08
      4       7   30374  152890   2.19
      8      15   19437  207742   2.98
     16      31   15173  331559   4.76
     32      63   14099  636086   9.13
     64     127   16804 1497220  21.48
    128     255    8390 1470464  21.10
    256     511    3003 1033383  14.83
    512    1023     810  551813   7.92
   1024    2047     258  370811   5.32
   2048    4095     101  282202   4.05
   4096    8191      27  145550   2.09
xfs_db: frag -d
actual 45966, ideal 12398, fragmentation factor 73.03%
xfs_db: frag -f
actual 2104856, ideal 2100484, fragmentation factor 0.21%
```

xfs_fsr

- Simple defragmentation tool that
  - Searched for files that are fragmented
  - Creates a tempory inode
  - Asks the filesystem to create new extents for the temporary inode
  - If the new extents are less fragmented it copies the data in original file to the new extents
  - The temporary inode is then renamed to replace the original file

- Fsr makes no consideration for the used and free space within its allocation group and does not rearrange files to create larger contiguous free space
- So fsr may fragment free space over a period of time

sgi®

sgi®