

Tuning and Troubleshooting Filesystems

By: Micah Altman, (Copyright 1994-7)

Last Revision May 20, 1998

Monitor Performance

By monitoring the you can catch small problems before they get big, and can help you to catch intermittent problems that may be hard to reproduce.

- *Run periodic tests and benchmarks* to check that system services are working. Keep a record of previous benchmarks to track system performance. However, avoid doing system intensive benchmarks during periods of high system use.
- *Gather performance statistics with tools like sar (-u,-b,-d,-c) and fsr.* Use `ssusage` (IRIX) or `timex -s` to benchmark individual programs (note that `timex` statistics can be misleading as they do not isolate activity for the individual program from system activity). Performance and error statistics can point out problems or bottlenecks in particular subsystems and at particular times. Is the CPU waiting on filesystem or physical I/O? If so, you have a disk/tape/cd bottleneck. Use `sar -d` to show which device is busy. Even if the CPU is not waiting on I/O if you see a lot of block reads/writes with `sar -b` you may have mild bottleneck.
- *Keep good records* of setup, performance and changes. Knowing what has changed on a network can speed the troubleshooting process by pointing out areas to investigate.
- *Check log files*, many network problems can be quickly diagnosed by a thorough reading of the log files. A good place to start looking for log messages is the `console` device and the `SYSLOG` file.

- Check the `SYSLOG` for such problems as unrecoverable media errors, scsi bus resets, and out of disk space errors

- Use `fx` to look at bad-block lists — have they grown? You can use the `label/set/parameters` option in `fx -x` to turn on bad-block notification.

Disk Troubleshooting

- **KEEP BACKUPS** of your filesystem data. It is much easier to restore from backup than to try to recover files from a bad disk. Check your backups to make sure that they work, and keep some copies off-site.
- *Running out of disk space* is the most common problem associated with disks. When a disk runs out of space, many unexpected programs fail because they cannot create temporary files. Using a

filesystem that is “almost” full for extended periods of time is likely to lead to filesystem fragmentation (see below) and reduce performance.

- Use `cron` and `find` to remove old temporary files, core files, system panics, auditing and log files, files created in `/dev` by root writes to non-existent devices (etc.) — but make sure that your scripts don’t remove anything important.
- Archive old files (especially auditing logs) to secondary storage.
- Create separate partitions for disk intensive services like news-server directories and auditing logs.
- Consider using disk quotas for users and some system services — especially for disk intensive system services like net-news servers and mail programs that may unexpectedly go off-kilter, or that are prone to denial-of-service attacks from the outside. (Note: XFS support for quotas came as a patch for 6.2. With this patch, however, XFS quotas are more flexible than `efs` quotas in two important ways: XFS keeps quota information in the log subvolume, XFS can use quotas for accounting only, as well as to set limits.)
- *Inodes* are the basic index structures used to keep track of files and directories on the disk. Know how to manipulate them:
 - If you run out of filesystem inodes you will not be able to create new files or directories. You can use `df -i` to see how many inodes are available. If you are creating a filesystem and you think that it will contain an unusual number of files, use `mkfs -n` to increase the number of inodes. (The XFS filesystem creates these dynamically, up to a maximum percentage (default 25%) of the filesystem, so this is not a problem when using it. In the extremely unusual circumstance that you need more than 25% of the filesystem for inode data, you must reconfigure with `mkfs`.)
 - If you run out of in-memory inodes programs will not be able to open any more files, until some files are closed. Use `systemd` to adjust the in-memory inode table; also consider adjusting the maximum number of file-descriptors per program.
 - You can use `find -inum` to look for a file with a particular inode value, you can use `ncheck` to find the file that corresponds to a particular inode.
 - Having trouble deleting a file? If all else fails, use `clri` (EFS only) to nuke the file’s inode.
- *Boot disks* contain a number of special programs (`fx`, `SASH`) in the volume header, if you are making a boot disk from scratch, copy these programs from a working boot disk using `dvhtool -v get/create`.

- *Bad blocks* occur when disk media ages and becomes unreliable:
 - Sometimes reformatting can temporarily cause bad-blocks to disappear, but they usually come back. A better way of managing bad blocks is to use `fx` to exercise the disk - which looks for bad blocks and adds them to the bad block list. Use the `butterfly` option to spread out these tests, this will stress the disk head and find bad-blocks that you might otherwise miss.
 - The parameters that control whether bad blocks are corrected automatically, whether the an error is reported for bad blocks, and the number of disk-retries are all configurable using the `label/set/param` option in `fx -x`.
- *Logical Volumes (lv's)* raise special issues.
 - Some versions of the PROM/sash boot-up process does not recognize lv's, which can complicate some recovery. You can make xfs lv's the root filesystem but you must follow special procedures in the man page.
 - Information about the structure of the logical volume is kept in the disk label, if this becomes corrupt, or there is "old" information in it, first flush any in-memory xlv information by running `xlvm_shutdown`. Then, if you want to start from scratch, delete the xlv information in the disk label with the `delete object` or (more drastic) `delete all_labels` commands in `xlvm_mgr -x` or (more drastic) erase the disk label with `fx`. Delete the device files from `/dev/xlv` or `/dev/(r)disk/xlv`. Otherwise, use the `-f` option to `lvinit` or the `-f` option within `xlvm_make` (or the `-force` option to the `ve` subcommand in `xlvm_make`) to construct the volume. You can update things from within `xlvm_mgr` without exiting by using the `reset` command.
 - If you change the name of your host, or the location of disk-controllers and disks, you may confuse the logical volume. You can use the `change nodename` command from `xlvm_mgr -x` for the first case. For the other cases, you can work around the changes by creating extra plexes and detaching the originals.
 - If you stripe your logical volumes, the failure of any single disk is likely to irreparably damage all files on the filesystem.
- *File system recovery* is not for the faint of heart, but it may be your only option if backups are unavailable.
 - EFS: Use `fsck` to automatically check and repair your disk. If `fsck` reports problems use the `-l` option to recover files to a different disk. Normally `fsck` will put lost files in the `lost+found` directory. You could also use `fsdb` to try to recover the files yourself, but it

is not recommended. If `fsck` cannot locate the superblock (the master index structure) you may be able to rebuild it using `mkfs -r`. This is a desperation option.

- MAC/DOS: Use `fsck` to recover MAC or DOS disks.
- XFS: The `xfs_check` program will check XFS filesystems but does not have any repair options. If this core dumps, you probably have a corrupted filesystem. Look for `xfsdb` and `xfs_repair` in 6.3 releases.
- If repair programs halt in the middle, try mounting read only (`-r`) and backing up the disk.
- Logical volumes: If you use a striped logical volume, and the disk fails, file recovery is quite unlikely. On a non-striped logical volume, delete the `xlvs`, replace the failed disk, then build the `xlvs` again, using *exactly the same parameters as the original*. Run `xfs_repair` on the new volume and you should be able to recover the files not residing on the failed disk.
- If all else fails, you can use `dd` to dump raw disk data to another disk (make sure there is room for it) or through `grep` or other filters, in the hopes of recovering some file data. (In 6.5 and beyond, use `dd conv=ignore`, to have it do all the hard work itself, and also write zero's on the destination blocks that can't be read from the source.

Maximizing Performance

- *Efficient i/o programming* is the most important component of performance, but also the component over which we typically have the least control. If you can specify how disk i/o is performed then the following techniques will improve your performance:
 - For most programs, use asynchronous reads and writes where possible — this will increase performance by allowing your program to perform other work while the system fetches data from disk, and will allow the system to cache filesystem data for higher performance.
 - Read data sequentially instead of jumping around in files — this allows the kernel to fetch data ahead of when you need it, so that it is there instantly when you ask for it.
 - Make the most use of the data you have – the more a program can work in one place in the dataset before moving to the next, the more likely it is to benefit from caching.
 - Perform reads and writes in large, memory-aligned chunks. Blocks of 64k (or multiples) are (currently) the most efficient because they take advantage of the natural transfer size of the disk controller. (Use `par` or `sar -c` to estimate the blocksize used by “mystery” programs)
 - If your program is trying to stream data to/from disk as fast as possible (e.g. live video) use the XFS-XLV filesystem’s Guaranteed Rate I/O programming features.

- *Caching*: normally the kernel i/o routines will cache data in RAM memory so that it can be accessed quickly the next time it is needed.
 - Keep some memory free, if there is no memory available on the system, the kernel will not be able to perform file-system caching.
 - Keep file-system data around: the daemon `bdflush` (`xfsd` for XFS filesystems) runs periodically to flush file-system data. By using `systemd` to control the parameters to `bdflush` you can improve the effects of caching by retaining file-system data. Be aware that this makes it more likely that you will lose data if there is a power outage.
 - In 6.2, two daemons similar to `bdflush` are introduced: `vfs_sync` operates on file-system metadata and is controlled by the `vfs_syncr` variable, `pd_flush` operates on memory mapped pages.
 - Increase file-system related kernel tables: if your system has an unusually high level of file-system activity consider increasing the path-name cache, the file-buffer header table, and the vnode free ratio with `systemd`. If you are using disk-quotas, consider increasing `ndquot`.
- *Simplify the filesystem*: broad and deep directory structures make opening individual files more expensive. If you are opening lots of files, simplify the directory structure. If you are opening a lot of files in one directory use `pwd` (change working directory) and relative addressing, rather than full-path names.
- *Use Data in Blocks* -The XFS file system allows you to configure the size of basic blocks and inodes on the filesystem (configure with `mkfs`, `xfsgrowfs -n` will show you the current blocksize and inode size of the filesystem.) , and IRIX allows you to configure page size (with `systemd`). Applications that grab data in large chunks, and that process it intensively before the next chunk are most likely to perform optimally, and applications that stream data contiguously to or from disk will benefit from large basic block (and page) sizes to maximize economies of scale in transfers from disk to memory. Block and page sizes should be multiples of each other and of the disk controller transfer size. Applications that access large files will benefit from large inodes. Large blocks and inodes may lead to wasted space on filesystems which contain lots of small files (like news spools).
- *Check for disk fragmentation*: disk fragmentation can lessen throughput. Use `fsr` (for EFS filesystems only) to defragment your disk. Disk fragmentation is much more likely to occur when your disk is kept more than 95% full. There is no defragmentation utility for XFS, but under 6.5 the `xfldb -c frag` will report fragmentation statistics.

- *Consider using striped logical volumes* if you are trying to maximize the performance for large sequential disk transfers (or possibly for many very small simultaneous entries). Striping spreads your files across several disks, allowing you to parallelize disk transfers.
 - Only stripe across disks of the same speed - otherwise performance will be limited by the speed of the slowest disk.
 - Make sure not to create a bottleneck in the controller or the bus — placing many striped disks on a single controller will just cause a performance bottleneck in the controller. Arrange disks in your logical volume so that you spread accesses evenly across controllers.
 - The stripe size should be an exact multiple of the block size, and divide evenly into the expected i/o size of your application (making the stripe size an even multiple of page size and the transfer size of the controller is also a good idea).
 - Striped logical volumes in EFS are not as fast as those using XFS. Use an external log for maximum speed. Plexing in XFS tends to slightly increase read performance and to decrease write performance. XFS's "Realtime" volume are even faster but require an intimate knowledge of the application programs and i/o programming with GRIIO (guaranteed rate input output) calls — see the XFS manual for details on setting up a Realtime volume.
- *Plexing* - Typically plexes will not affect performance dramatically if you use disks of the same type for each plex.
 - Reads will complete somewhat faster (reads finish when any plex finishes) but writes will be slower (writes finish when all plexes finish).
 - Plex revives can take a lot of time, but happen in the background. To speed up the plex revive, increase the i/o chunk used when reviving the plexes. You can adjust the `xlvs_plexd(1m)` revive parameter size (-b) and pause (-w) between revive chunks. See the man page.
- *Hardware matters* – fast and wide SCSI devices can outperform thin and slow devices. Check the seek time and transfer rates of your disk, and the performance ratings on your controllers and busses.