SEE TEXT ONLY

svn UPDATE

by Steven Kreuzer

What advantages does FreeBSD have over Linux? That's a question I get asked quite a bit, and it is hard to answer because FreeBSD does a lot of things really well. Where do you start? You could ramble on about the robust networking stack or cutting-edge technology such as DTrace and Capsicum for hours on end. However, I would argue that the area where FreeBSD really shines is storage. Not only are developers spending quite a bit of time making sure that the 1s and 0s you save to the disk get written in the correct order, but that it is doing so as quickly as possible. If that wasn't enough, they are also making sure that those same 1s and 0s get read back in the correct order as quickly as possible. Whether you are just archiving family photos on your laptop or serving a multi petabyte ZFS volume to thousands of clients on the network, FreeBSD provides a robust and reliable platform to meet your storage needs.

Provide a sysctl to force synchronous initialization of inode blocks. https://svnweb.freebsd.org/changeset/base/326731

FFS performs asynchronous inode initialization using a barrier write to ensure that the inode block is written before the corresponding cylinder group header update. Some GEOMs do not appear to handle BIO_ORDERED correctly, meaning that the barrier write may not work as intended. The sysctl allows one to work around this problem at the cost of expensive file creation on new filesystems.

Support mounted boot partitions in the installer. https://svnweb.freebsd.org/changeset/base/326674

This allows the platform layer, for example, to specify that the EFI boot partition should be mounted at /efi and formatted normally with newfs msdos rather than splatted to from /boot/boot1.efifat.

zfs_write: fix problem with writes appearing to succeed when over quota. https://svnweb.freebsd.org/changeset/base/326070

The problem happens when the writes have offsets and sizes aligned with a filesystem's recordsize (maximum block size). In this scenario dmu_tx_assign() would fail because of being over the quota, but the uio would already be modified in the code path where we copy data from the uio into a borrowed ARC buffer. That makes an appearance of a partial write, so zfs_write() would return success and the uio would be modified consistently with writing a single block. That bug can result in a data loss because the writes over the quota would appear to succeed while the actual data is being discarded.

This commit fixes the bug by ensuring that the uio is not changed until after all error checks are done. To achieve that, the code now uses uiocopy() + uioskip() as in the original illumos design. We can do that now that uiocopy() has been updated in r326067 to use vn_io_fault_uiomove().

Avoid holding the process in uread() and uwrite(). https://svnweb.freebsd.org/changeset/base/325887

In general, higher-level code will atomically verify that the process is not exiting and hold the process. In one case, we were using uwrite() to copy a probed instruction to a per-thread scratch space block, but copyout() can be used for this purpose instead; this change effectively reverts r227291.

Optimize telldir(3). https://svnweb.freebsd.org/changeset/base/326640

Currently each call to telldir() requires a malloc and adds an entry to a linked list which must be traversed on future telldir(), seekdir(), closedir(), and readdir() calls. Applications that call telldir() for every directory entry incur O(n^2) behavior in readdir() and O(n) in telldir() and closedir().

This optimization eliminates the malloc() and linked list in most cases by packing the relevant information into a single long representation. On 64-bit architectures msdosfs, NFS, tmpfs, UFS, and

ZFS can all use the packed representation. On 32-bit architectures, msdosfs, NFS, and UFS can use the packed representation, but ZFS and tmpfs can only use it for about the first 128 files per directory. Memory savings is about 50 bytes per telldir(3) call. Speedup for telldir()-heavy directory traversals is about 20-30x for one million files per directory.

Tweak seekdir, telldir, and readdir so that when there are deletes, seeks to the last location saved will work. https://svnweb.freebsd.org/changeset/base/282485

This is needed for Samba to be able to correctly handle delete requests from windows. This does not completely fix seekdir when deletes are present but fixes the worst of the problems. The real solution must involve some changes to the API for eh VFS and getdirentries(2).

Avoid the overhead of acquiring a lock in nfsrv_checkgetattr() when there are no write delegations issued. https://svnweb.freebsd.org/changeset/base/326544

manu@ reported on the freebsd-current@ mailing list that there was a significant performance hit in nfsrv_checkgetattr() caused by the acquisition/release of a state lock, even when there were no write delegations issued. This patch adds a count of outstanding issued write delegations to the NFSv4 server. This count allows nfsrv_checkgetattr() to return without acquiring any lock when the count is 0, avoiding the performance hit for the case where no write delegations are issued.

zfsd should be able to online an L2ARC that disappears and returns. https://svnweb.freebsd.org/changeset/base/325011

Previously, this didn't work because L2ARC devices' labels don't contain pool GUIDs. Modify zfsd so that the pool GUID won't be required.

Fix zpool_read_all_labels when vfs.aio.enable_unsafe=0. https://svnweb.freebsd.org/changeset/base/324991

Previously, zpool_read_all_labels was trying to do 256KB reads, which are greater than the default MAXPHYS, and, therefore, must go through the slow, unsafe AIO path. Shrink these reads to 112KB so they can use the safe, fast AIO path instead.

Fix the error message when creating a zpool on a too-small device. https://svnweb.freebsd.org/changeset/base/324940

Don't check for SPA_MINDEVSIZE in vdev_geom_attach when opening by path. It's redundant with the check in vdev_open, and failing to attach here results in the wrong error message being printed.

Add vfs_zfs.abd_chunk_size tunable. https://svnweb.freebsd.org/changeset/base/323797

It is reported that the default value of 4KB results in a substantial memory use overhead (at least, on some configurations). Using 1KB seems to reduce the overhead significantly.

Add sysctls for arc shrinking and growing values. https://svnweb.freebsd.org/changeset/base/323051

The default value for arc_no_grow_shift may not be optimal when using several GiB ARC. Expose it via sysctl allows users to tune it easily. Also expose arc_grow_retry via sysctl for the same reason. The default value of 60s might, in case of intensive load, be too long.

msdosfs(5): Reflect READONLY attribute in file mode. https://svnweb.freebsd.org/changeset/base/326031

Msdosfs allows setting READONLY by clearing the owner write bit of the file mode. In msdosfs_getattr, intuitively reflect that READONLY attributes to userspace in the file mode.

Use taskqueue(9) to do writes/commits to mirrored DSs concurrently. https://svnweb.freebsd.org/changeset/base/324676

When the NFSv4.1 pNFS client is using a Flexible File Layout specifying mirrored Data Servers, it must do the writes and commits to all mirrors. This change modifies the client to use a taskqueue to perform these writes and commits concurrently. The number of threads can't be changed for taskqueue(9), so it is set to 4 * mp_ncpus by default, but this can be overridden by setting the sysctl vfs.nfs.pnfsiothreads.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retrocomputing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.