Improving the Performance of fsck in FreeBSD

MARSHALL KIRK MCKUSICK

Dr. Marshall Kirk McKusick writes books and articles, teaches classes on UNIX- and BSD-related subjects, and provides expert-witness testimony on software patent, trade secret, and copyright issues, particularly those related to operating systems and filesystems. His work with Unix and BSD development spans over four decades. It begins with his first paper on the implementation of Berkeley Pascal in 1979, goes on to his pioneering work in the '80s on the BSD Fast File System, the BSD virtual memory system, the final release of 4.4BSD-Lite from the UC Berkeley Computer Systems Research Group, and carries on with his work on FreeBSD.
mckusick@mckusick.com

While listening to the presentation of the first paper at FAST '13, "ffsck: The Fast File System Checker" [1], I immediately wondered whether I could implement some of the ideas in FreeBSD. The researchers' goal was to reorganize the Linux ext3 filesystem and to rewrite its filesystem checker so that a complete check of the filesystem could be done more quickly. With the addition of a couple of hundred lines of code, I was able to implement both the improvements to fsck and the layout policy in the FreeBSD filesystem (FFS).

Although the thrust of the paper was to make changes to the layout of the filesystem to enable fsck to run more quickly, some of the changes resulted in a reduction in performance of the filesystem. As I am unwilling to accept a reduction in filesystem performance solely for the purpose of speeding up fsck, I chose to consider only on the subset of their changes that improve both.

Implementation

The paper describes changes that the researchers made to the on-disk layout of the filesystem. Getting folks to change to a different filesystem format that is incompatible with the existing filesystem format is difficult. So, in my implementation, I was not willing to change the file-system format beyond using one of the spare fields in the superblock to tune the layout policy. Even with these limitations, I was able to get an impressive improvement in fsck's running time and some small improvements in filesystem performance.

In FFS (the Fast FileSystem), the disk space is broken up into groups of contiguous blocks called cylinder groups similar to the ext3 block groups. The first block of each cylinder group contains the cylinder group descriptor that includes a map showing the free and allocated blocks and a map showing the free and allocated inodes in that cylinder group. Following the cylinder group descriptor are blocks that contain the metadata (inodes) for the files in that cylinder group. The organization of an inode is shown in Figure 1. The remainder of the cyl-inder group is made up of blocks that contain the indirect blocks and data blocks for the files and directories contained in the filesystem. An inode may reference blocks in one or more cylinder groups in the filesystem, although the policy is that small files have their blocks allocated in the same cylinder group in which the inode resides. For details, see Chapter 8 of McKusick & Neville-Neil [2].

The key idea in the paper [1] is to reserve a small area in each cylinder group immediately following the inode blocks for the use of metadata, specifically indirect blocks and directory contents. It requires that metadata be allocated in this area and does not allow data blocks to be allocated in this area. Thus, the paper has a long discussion of how to size this area. If it is improperly sized, the filesystem will report as being full when it in fact still has plenty of available space since it reports a filesystem full error when either the metadata area or the non-metadata area fills up.

The FFS separates the allocation of data blocks and inodes into two distinct layers: policy and implementation. The policy layer is responsible for picking what it views as the ideal place to

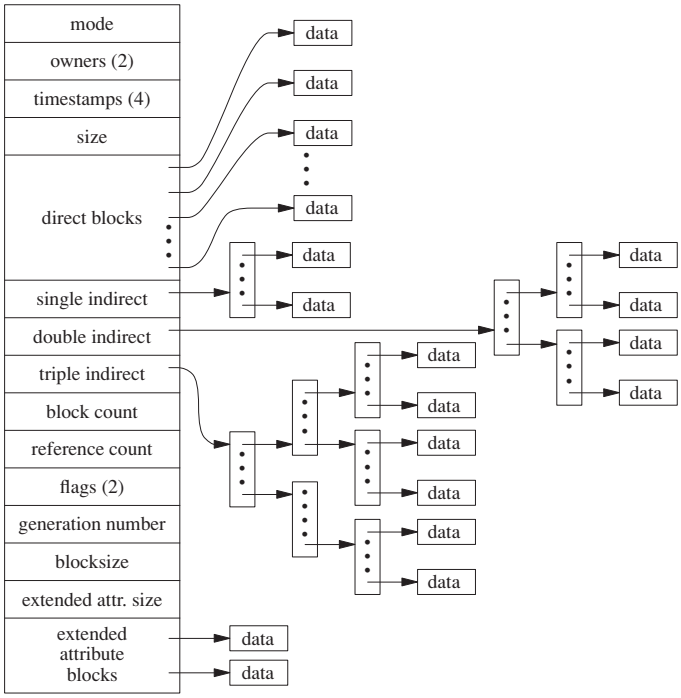

Figure 1. The structure of an inode

allocate the inode or the data block. For example, when asked to allocate a block for a file, the policy layer will usually ask for the block that immediately follows the previously allocated block.

The implementation layer is responsible for managing the allocation bitmaps and ensuring that resources do not get double allocated. Thus, the policy layer does not have to worry about requesting an already allocated block. If the implementation layer finds that a requested block is already allocated, it simply scans through the map to find the closest available free block. The result of this separation is that once the implementation layer is working properly, filesystem designers are free to try out whatever hare-brained policy ideas that they want without fear of corrupting the filesystem. In the case of FFS, the implementation layer was written and debugged in 1982 and has not been changed since. Further refinements to the filesystem have been done at the policy layer.

Following these design principles, I chose not to change the filesystem layout or the implementation layer. Instead I chose to implement it entirely as a new policy. Specifically, the new policy is to hold about the first 4% of the data blocks in each cylinder group for use of metadata. The policy routines preferentially place metadata in the metadata area and everything else in the blocks that follow the metadata area. In my implementation, the size of the metadata area does not matter as it is just used as a hint by the policy routines. If the metadata area fills up, then the metadata just gets put in the regular blocks area and vice versa.

And this decision happens on a cylinder group by cylinder group basis (e.g., some cylinder groups can overflow their metadata area whereas others do not overflow it). For filesystem performance, having the metadata in the same cylinder group as its inode is usually better than pushing it to the metadata area of another cylinder group as is done by the design in the paper.

Another area where I chose to take a different approach than the paper is in the allocation policy for the first indirect block of the file. The BSD fast filesystem tries to place the first (single) indirect block inline with the file data (e.g., it tries to lay out the first 12 direct blocks contiguously followed immediately by the indirect block followed immediately by the data blocks referenced from the indirect block). One of the performance slowdowns in the paper occurs for files that spill into only the first part of their first indirect block. The slowdown comes from moving this first indirect block to the metadata area, thus causing two extra seeks when reading it. To avoid this slowdown, I do not change the layout of the first indirect (leaving it inline). Only the second and third level indirects along with the indirects that they reference are moved to the metadata area. The nearly contiguous allocation of this metadata close to the inode that references it noticeably improves the random access time to the file as well as speeding up the running time of fsck. Also, as noted in the paper, the disk track cache is frequently filled with much of a file's metadata when the second level indirect block is read, thus often speeding up even the sequential reading time for the file; however, in limited testing I did not see statistically significant differences in sequential reading times.

Putting the contents of directories in the metadata area gives a similar speedup to directory tree traversal because the data is a short seek away from where the directory inode was read and may already be in the disk's track cache from other directory reads done in its cylinder group.

The final observation that I plucked from the paper specifically for speeding up fsck is to save an in-memory copy of the cylinder groups during pass1 so as not to need to re-read them in pass5. This nearly doubles the memory footprint of fsck, so if memory runs short (e.g., its mallocs begin to fail) this cache is released as needed to make room for other allocations.

Results

I have been testing on an Intel Quad-core CPU running at 2.83 GHz with 2 Gb of memory and a 2 Tb Western Digital 7200 rpm testing disk running FreeBSD 8.3-STABLE (Subversion revision r246915M). Filesystems are created with their default settings: 16 K blocks, 2 K fragments, soft updates, and 4% of the data blocks held for metadata. For these tests, the filesystem is 75% full mostly populated with big files (to exaggerate the metadata effects). In each case a new filesystem was created and all the data copied into it so that the new layout could have maximal

effect. There are few files and hence little directory information, so the benefit to the running time for directories is minimal in these tests. I am currently running tests on a more conventionally populated filesystem.

Fsck times are also better as the filesystem has not been aged; however, aging effects in the FFS filesystem tend to be a lot less noticeable than in others because of its use of dynamic block reallocation. Notably, the Harvard folks found that I/O performance dropped off by only about 10% after 10 months of simulated aging [3]. Also, fsck times are low because of the small number of files in the filesystem and hence the smaller number of inodes needing to be inspected. Finally, a technique similar to the metadata compression discussion in the Ma, et al paper has been in use in fsck for the directory metadata since 1988, which cuts down on running time.

Executive summary on running time of fsck:

- Baseline before any changes: 284 seconds (4 min 44 sec)
- Storing second and third level metadata (and their referenced indirect blocks) but not first indirect block in the metadata area: 135 seconds (2 min 15 sec)
- Adding directory data blocks to metadata area: 134 seconds (2 min 14 sec)
- Caching cylinder group blocks in pass1 to avoid the need to read them in pass5: 84 seconds (1 min 24 sec)

In Appendix 1 [4] are the summary statistics for each run. I/O listed as "Double Level Indirect" includes all double-indirect blocks referenced from inodes and all the single level indirect blocks below them. Similarly, "Triple Level Indirect" includes all triple-indirect blocks referenced from inodes and all the single and double level indirect blocks below them. The key observation is that whereas the number of I/Os of each type of data remain similar from run to run, the percentage of time for reading the metadata has dropped dramatically.

I ran just a few tests on the speed with which data could be read from or written to files. Random read times improved a bit. The remaining tests were not statistically significantly different. More thorough tests would need to be run to get a reasonable idea of whether it makes any difference; first results imply no degradation and some hints at improvement.

Conclusions

This work has once again shown the power of separating the filesystem layout policy routines from the implementation routines. I was so excited by the possibilities presented by the FAST '13 paper that I skipped lunch after hearing the presentation so I could try implementing it in FFS. By the time the 90 minute lunch break was over, I had fully written the 100 lines of changes (half of which were comments) to the block layout policy routine to implement the reserved metadata area. And I had no fears of bringing it up on my primary server to test it out because I knew that at worst I would get some badly laid out files; certainly I was not running the risk of corrupting my filesystems.

By retaining the same on-disk format, I did not need to make any changes to fsck. The stock fsck just ran faster because of the new layout of metadata. I did need to make about 100 lines of changes to fsck to add the caching of cylinder groups between pass1 and pass5; however, that was a trivial change and one that will provide equal improvement whether or not the new filesystem layout is in use. The vast majority of my time has been spent measuring the effects of the changes and writing this paper. Having spent time writing or tuning fsck for the past 30 years, I never would have guessed that so much improvement in running time could come from fsck for so little effort.

The lesson to be learned is that separating policy from implementation is an important design principle when architecting software systems, especially when they are mission-critical systems. The policy layer allows new ideas to be implemented and tested quickly. Once validated, those ideas can be deployed without danger of compromising the integrity of the system.

I commend the authors of the paper for their work. Unfortunately the filesystem on which they worked is not separated into policy and implementation layers, so they had to make several thousand lines of changes in areas where bugs would compromise the filesystem integrity. The monolithic architecture led to a great deal more effort on their part than would otherwise have been necessary. Finally, the scope of the change and the possibility of destabilizing a production filesystem will make it far more difficult for them to get their changes accepted back into the mainline code base.

References

[1] A. Ma, C. Dragga, A. Arpaci-Dusseau, & R. Arpaci-Dusseau, "ffsck: The Fast File System Checker," 11th USENIX Conference on File and Storage Technologies (FAST '13), http://www.usenix.org/conference/fast13/ffsck-fast-file-system-checker (February 2013).

[2] M. K. McKusick & G. V. Neville-Neil, The Design and Implementation of the FreeBSD Operating System, Addison-Wesley, Reading, MA (2005).

[3] K. Smith & M. Seltzer, "A Comparison of FFS Disk Allocation Algorithms," Winter USENIX Conference, pp. 15-25, http://www.eecs.harvard.edu/margo/papers/usenix96-ffs (January 1996).

[4] Appendix 1: http://www.usenix.org/publications/login/april-2013-volume-38-number-2.

ORACLE®

LABS

Research Partnerships

The Mission of Oracle Labs is to identify, explore, and transfer new technologies that have the potential to substantially improve Oracle's business. Oracle Labs researchers look for novel approaches and methodologies, often taking on projects that are high risk, uncertain, or difficult to tackle within a product development organization. Oracle Labs research is focused on real-world outcomes: our researchers aim to develop technologies that will someday play a significant role in the evolution of technology and society. We maintain a balanced research portfolio, including exploratory research, directed research, consulting and product incubation.

The External Research Office at Oracle Labs invests in research collaborations that narrow technology gaps, explore and expand new technologies. Oracle Labs hosts undergraduate and graduate student interns, postdocs, and visiting professors to partner with leading researchers in onsite collaborations. Oracle Labs also funds off-site research through partnerships with research institutions worldwide. In the past two years, Oracle Labs funded 50 universities and 90 students worldwide. Last year alone, Oracle Labs hosted 63 interns.

Areas of Research

VLSI Circuit Design
Hardware / Software Co-design
Query Processing on Extreme Scale-out Architectures
Processor Design
Integrated Circuit Packaging
Domain-Specific Language Design and Implementation
Managed Language VM technologies
Silicon Photonics
Information Retrieval and Machine Learning
Operating Systems in the Cloud
HW Application Accelerators in the Network
Persistent Programming Languages
Program Analysis for Correctness & Vulnerability Detection
Graph Analytics
Scalable Concurrency

Locations

Based at Oracle headquarters in Redwood Shores, CA, Oracle Labs also has research centers in Boston, MA, Austin, TX, San Diego, CA, Linz, Austria, Cambridge, UK, Vancouver, CA, and Brisbane, Australia. Senior individual researchers work from remote locations all over the world, from Germany to New Zealand. The geographic spread allows Oracle Labs to take advantage of a tremendous pool of scientific and engineering talent and enables Labs researchers to collaborate with colleagues from a wide range of industries and universities.

For more information on how to establish a research collaboration, visit https://labs.oracle.com, or contact Marie-Therese Ellis at marie-therese.ellis-house@oracle.com.