A Critique of Seltzer's 1993 USENIX Paper

John Ousterhout / john.ousterhout@eng.sun.com

The paper "An Implementation of a Log-Structured File System For UNIX", by Margo Seltzer et al., appears in the Winter 1993 USENIX Conference Proceedings (pages 307-326). The paper describes a new implementation of a log-structured file system (LFS) in the BSD UNIX kernel, then presents a performance comparison between three file systems: the new LFS implementation; FFS, the traditional BSD file system; and EFS, a modification of FFS that supports extent-based allocation.

The BSD implementation of LFS differs from the original Sprite implementation in several ways, some of which are definite improvements. The most notable improvement in BSD-LFS is a new approach to cleaning that uses optimistic concurrency control and a user-level cleaner; I believe that this is a much better way to do cleaning than the approach used in Sprite, and we adopted Seltzer's approach in Sprite's Zebra file system.

However, several of the differences between BSD-LFS and the Sprite implementation are steps backwards (see below for details). More importantly, the performance measurements in Section 5 of the paper are seriously flawed. The comparisons between BSD-LFS, FFS, and EFS suffer from three general problems: a bad implementation of BSD-LFS, a poor choice of benchmarks, and a poor analysis of the benchmarks. Combined together, these problems invalidate many of the paper's conclusions; LFS is a much better file system architecture than this paper suggests.

Fortunately, a newer paper by Seltzer in the Winter '95 USENIX, "File System Logging Versus Clustering: A Performance Comparison", corrects many of the flaws in the 1993 paper. Although the newer paper also has flaws, its conclusions are much closer to the truth. Thus I recommend that the quantitative results of the 1993 paper be disregarded in favor of those in the 1995 paper.

The sections below elaborate on the three general problems with Seltzer's implementation and measurements.

Poor BSD-LFS Implementation

At the time this paper was written the BSD-LFS implementation contained a number of flaws that affected its performance:

- The system did not implement fragments, so the smallest block size was 4KB or 8KB, compared to 512 or 1024 bytes for EFS and FFS. This resulted in an unnecessary 4-8x reduction in LFS performance for small files.
- BSD-LFS contained a bug causing it to layout segments backwards on disk. This damages read performance during sequential reads by wasting almost a full disk rotation between each block of a file and the next block. The performance penalty is most noticeable for medium-sized files.
- BSD-LFS stores the access time in inodes, rather than putting it in the inode map as in LFS. This means that the inode must be rewritten each time the file is read, so the inode tends to migrate away from the file on disk, causing long seeks during read accesses.
- BSD-LFS flushes indirect blocks and inodes to disk unnecessarily, resulting in excess writes and increased cleaner overhead (this problem is described in more detail in my critique of the 1995 USENIX paper).

The first two of these problems were fixed for Seltzer's 1995 paper, while the other two bugs are still present in the system. I could not find a mention of these problems in the 1993 paper, yet they invalidate much of the data in Figures 9 and 12 and affect the other measurements as well.

Poor Benchmark Choice

The goal in choosing benchmarks should be to find ones that reflect as closely as possible the real-world usage of a system. This is a difficult task for file systems and I don't know of any "perfect" file system benchmarks, but the selection for the paper is particularly bad:

- The paper does not evaluate small-file performance at all, even though most files in real-world applications are small. For example, file sizes less than 100 Kbytes are barely visible at the left edges of Figures 9 and 12. The software development workload uses small files, but it is mostly CPU bound so it doesn't demonstrate differences in file system performance for small files. LFS is at its best for small files, so by omitting small-file benchmarks the paper biases against LFS.
- Section 5.2.1, on raw file system performance, is based on a most unusual benchmark that overwrites existing file data rather creating new data, and does the writes synchronously. Furthermore, it appears not to include the time to actually open the file being read or written, and it apparently accesses a single file repeatedly rather than using a collection of files. It is difficult to imagine a real application that would generate this workload, and the benchmark certainly doesn't reflect typical behavior. The data in Figures 9 and 12, where EFS appears to be uniformly better than LFS, is an artifact of the bad benchmark and the bugs in the BSD-LFS implementation. In the '95 USENIX paper some of the BSD-LFS bugs have been fixed and a different benchmark is used, which is much more representative of actual file system use; in that paper LFS performs substantially better than EFS or FFS.
- The Andrew benchmark used in Section 5.2.2 is mostly CPU bound, so it doesn't provide much information on file system performance. The multi-user version of the benchmark seems totally artificial to me, since a single user is unlikely to run multiple compilations simultaneously.
- All of the benchmarks use a file system cache of only 1 Mbyte. Considering that the machine had a total of 16 Mbytes of memory, this is an relatively small cache size, and it impacted the transaction processing results in Section 5.2.3. LFS was designed for systems that use caching agressively, so restricting the cache size biases against LFS. The 1995 USENIX paper used a more realistic cache size and as a result LFS performed much better relative to EFS in the transaction processing tests.

In discussions about the paper, Seltzer and one of her co-authors admitted that they intentionally chose benchmarks that were unfavorable to LFS and omitted benchmarks where LFS performed best. They justified this by claiming that the original LFS paper by Rosenblum and myself used benchmarks that favored LFS and they were trying to present the other side of the story. Although I disagree with this assessment of Rosenblum's and my paper, the proper response is to present a fair set of benchmarks, not an unfair set biased the other way.

Poor Analysis

Unfortunately, the analyses in the paper do not address the deficiencies in the BSD-LFS implementation and the benchmarks. As a result, readers are left with the conclusion that LFS is architecturally inferior to EFS. In addition, several important factors are omitted in the discussions of the benchmark. Here are some examples of problems:

- The paper is not clear enough in stating that the measurements in Figure 9 are entirely an artifact of a bad benchmark and bugs in the LFS implementation. There is some explanation in the text, but it is hard to follow and the figure caption contains no indication that the results are meaningless. Readers will remember the figure, which suggests a 1.5-2x advantage for EFS, not the fine print in the text, which states that in fact the two systems are equal.
- The explanation for Figures 9 and 12 doesn't discuss the impact of a 56-Kbyte I/O limit, which accounts for most of the performance differences at large transfer sizes.
- Figure 14 consists of a blow-up of the Andrew benchmark measurements, and the paper elaborates on this blow-up to "explain" the performance differences. In fact the differences at high degrees of multiprogramming are statistically insignificant; in a class project at Berkeley these measurements were repeated and it was found that the results vary enough from run to run to cover all of the differences in the figure.
- In the discussion of transaction performance, the paper does not satisfactorily explain the cleaning performance for LFS. For example, it measures only a single disk utilization so it isn't possible to see how

LFS performance varies with utilization.
- The descriptions of the benchmarks are not sufficient to reproduce the measurements. For example, it isn't mentioned in Section 5.2.1 that the same file is used over and over again, and Section 5.2.3 doesn't say what the disk utilization was for LFS.

Conclusion

Of all the performance numbers in the paper, only the single-user Andrew performance numbers are particularly useful. All of the other measurements are inadequate for one of the reasons given above. I recommend that readers ignore the measurements in this paper and use instead those in Seltzer's 1995 USENIX paper, which are better designed and reflect bug fixes in BSD-LFS.

The paper concludes that "FFS (with read and write clustering) provides comparable and sometimes superior performance to our LFS". This may have been true for certain benchmarks on the flawed version of BSD-LFS that existed when the paper was written, but it is not true for realistic benchmarks on a well-tuned LFS implementation. Seltzer's newer paper shows that in fact BSD-LFS is much faster than EFS over a wide variety of operating conditions (as much as an order of magnitude in places), and even at its worst it is only a few percent slower than EFS.