A Response to Seltzer's Response

This article is a response to Margo Seltzer's response to my critique of her 1995 USENIX paper Logging versus Clustering: A Performance Evaluation.

Here are the four summary paragraphs from Seltzer's response (bulleted and italicized), followed by my comments.

- *Ousterhout references "a number of flaws in the earlier paper" (the 1993 Usenix paper* ***A Log-Structured File System for UNIX****), but does not detail these flaws. To date, we are aware of no technical inaccuracies in the paper.*

This is a surprising comment, given the number and intensity of discussions we have had on this topic. The 1993 paper is riddled with flaws, including errors in the BSD-LFS implementation, errors in the choice of benchmarks, and errors and omissions in the presentation and analysis of the results. In fact, Seltzer personally fixed two major performance problems in BSD-LFS during the preparation of the 1995 paper, and her response web page describes a flaw in the 1993 paper's analysis. Click here for a more complete critique of the 1993 paper.

- *Ousterhout cannot explain the measurements in Section 4, stating that they are inconsistent with the simulated results in the Rosenblum/Ousterhout paper that appeared in the February 1992 ACM Transactions on Computer Systems. The simulated results in the TOCS paper are for a different algorithm and different parameters than used by either Sprite-LFS or BSD-LFS. This is explained in more detail below.*

In the detailed explanation, Seltzer offers new theories about what might account for the insensitivity of LFS cleaner performance to CPU utilization. However, none of these theories is backed up with measurements (the measurements given by Seltzer neither prove or disprove her theories). In any case they make it even clearer that the paper does not adequately explain what is going on. I stand by my suggestion that the paper's results (and Seltzer's latest theories) should be taken with a grain of salt until the performance can be tied quantitatively to specific architectural features of LFS.

- *The optimization, suggested by Ousterhout, to improve the transaction processing performance has not yet been implemented. This optimization affects the ability of the file system buffer cache to cache dirty data, and the evaluation of its impact is beyond the scope of the 1995 Usenix paper. We agree that this work will be interesting and encourage Ousterhout to pursue this research avenue.*

As I explained in my critique, the optimization need not affect the ability of the file system buffer cache to cache dirty data; the current approach represents a performance bug that can be fixed in a way that improves TP performance without adversely affecting other applications.

- *Ousterhout claims that the results in Section 5 are invalid. We present even more data below that demonstrates that the results in Section 5 are indicative of real-world performance.*

I'm delighted to see the additional measurements, because they validate my concerns and contradict Seltzer's conclusion above. The new data show that the paper erred by a factor of two in its estimates of performance degradation due to fragmentation. Seltzer's new estimates of performance degradation are 14% for reads and 24% for writes (this is the average of her reported numbers), whereas the average degradations reported in the paper were 6% for reads and 14% for writes. In the worst case, fragmentation degrades read performance by 33% and write performance by 47%.

What's most interesting about this is that the cost of fragmentation in FFS is comparable to the cost of cleaning in LFS. In Rosenblum's measurements of production LFS usage in Sprite, cleaning effectively added 20-60% to the cost of writes (but nothing to the cost of reads); in Seltzer's TP benchmark, which represents a pathological case for LFS, overall degradation due to cleaning was about 45%. However, the cleaner in LFS runs in the background so it may be possible to hide the cleaning costs by cleaning during idle periods; in FFS the fragmentation affects the file layout so there is no way to escape the overhead.

The Real Conclusions

If the original paper by Rosenblum and myself is combined with all of Seltzer's data, including both the data in the two USENIX papers and the data in her web response, I think it is reasonable to draw the following conclusions:

- It is possible to find particular benchmarks and system configurations where either file system dominates the other.
- Considering typical usage patterns but ignoring the costs of cleaning in LFS and fragmentation in FFS, LFS performance is almost never worse than FFS and often substantially better. In the best case for LFS (writes of small files) LFS is 4-10x faster; in the worst case (writes of large files) LFS is about 5% slower.
- Fragmentation can degrade actual FFS read performance by 15% and write performance by 25%, while cleaning costs effectively add 20-60% to the cost of writes in LFS without affecting the cost of reads. In the worst case, fragmentation degrades overall FFS performance by about 35-45% while cleaning degrades overall LFS performance by about 45%.
- LFS performance for transaction processing workloads ranges from about the same to about 10% worse than FFS, depending on the disk utilization.
- The BSD-LFS implementation is still quite immature relative to FFS, so its performance is likely to improve relative to FFS as optimizations like the one described above are implemented.

It's also important to remember some of the other advantages of LFS that weren't addressed in Seltzer's measurements, such as faster crash recovery and the ability to handle striped disks and network servers much more efficiently than FFS. Overall, the available data suggests that LFS is a much better file system than FFS.