Analyzing IO Amplification in Linux File Systems*

Jayashree Mohan
Department of Computer Science,
University of Texas at Austin
jaya@cs.utexas.edu

Rohan Kadekodi
Department of Computer Science,
University of Wisconsin Madison
kadekodirohan@gmail.com

Vijay Chidambaram
Department of Computer Science,
University of Texas at Austin
vijay@cs.utexas.edu

1 INTRODUCTION

File systems were developed to enable users to easily and efficiently store and retrieve data. Early file systems such as the Unix Fast File System [1] and ext2 [2] were simple file systems. To enable fast recovery from crashes, crash-consistency techniques such as journaling and copy-on-write were incorporated into file systems, resulting in file systems such as ext4 [3] and xfs [4]. Modern file systems such as btrfs [5] include features such as snapshots and checksums for data, making the file system even more complex.

While the new features and strong crash-consistency guarantees have enabled wider adoption of Linux file systems, it has resulted in the loss of a crucial aspect: efficiency. File systems now maintain a large number of data structures on storage, and both data and metadata paths are complex and involve updating several blocks on storage. In this paper, we ask the question: what is the IO cost of various Linux file-system data and metadata operations? What is the IO amplification of various operations on Linux file systems? While this question is receiving wide attention in the world of key-value stores [6, 7] and databases [8], this has been largely ignored in file systems. File systems have traditionally optimized for latency and overall throughput [9–12], and not on IO or space amplification.

We present the first systematic analysis of read, write, and space amplification in Linux file systems. Read amplification indicates the ratio of total read IO to user data respectively. For example, if the user wanted to read 4 KB, and the file system read 24 KB off storage to satisfy that request, the read amplification is 6×. Write amplification is defined similarly. Space amplification measures how efficiently the file system stores data: if the user writes 4 KB, and the file system consumes 40 KB on storage (including data and metadata), the space amplification is 10×.

We analyze five widely-used Linux file systems that occupy different points in the design space: ext2 (no crash consistency guarantees), ext4 (metadata journaling), XFS (metadata journaling), F2FS (log-structured file system), and btrfs (copy-on-write file system). We analyze the write IO and read IO resulting from various metadata operations, and the IO amplification arising from data operations. We also analyze these measures for two macro-benchmarks: compiling the Linux kernel, and Filebench varmail. We break down write IO cost by IO that was performed synchronously (during `fsync()`) and IO that was performed during delayed background checkpointing.

Measure	ext2	ext4	xfs	f2fs	btrfs
File Overwrite					
Write Amplification	2.00	4.00	2.00	2.66	32.65
Space Amplification	1.00	4.00	2.00	2.66	31.17
File Read (cold cache)					
Read Amplification	6.00	6.00	8.00	9.00	13.00
File Read (warm cache)					
Read Amplification	2.00	2.00	5.00	3.00	8.00

Table 1: Amplification for Data Operations. ***The table shows the read, write, and space amplification incurred by different file systems when reading and writing files.***

We find several interesting results. For data operations such as overwriting a file or appending to a file, there was significant write amplification (2–32×). .Small random reads resulted in a read amplification of 2–8×, even with a warm cache. Metadata operations such as directory creation or file rename result in significant storage IO: for example, a single file rename required 12–648 KB to be written to storage. Even though ext4 and xfs both implement metadata journaling, we find XFS significantly more efficient for file updates. Similarly, though F2FS and btrfs are implemented based on the log-structured approach (copy-on-write is a dual of the log-structured approach), we find F2FS to be significantly more efficient across all workloads. In fact, in all our experiments, btrfs was an outlier, producing the highest read, write, and space amplification. While this may partly arise from the new features of btrfs, the copy-on-write nature of btrfs is also part of the reason.

2 ANALYZING LINUX FILE SYSTEMS

We measure the read IO, write IO, and space consumed by different file-system operations.

Data Operations. First, we focus on data operations: file read, file overwrite, and file append. For such operations, it is easy to calculate write amplification, since the workload involves a fixed amount of user data. The results are presented in Table 1.

Metadata Operations. We now analyze the read and write IO (and space consumed) by different file-system operations like file create, directory create, and file rename. We have experimentally verified that the behavior of other metadata operations, such as file link,

APSys '17, September 2, 2017, Mumbai, India

ACM ISBN 978-1-4503-5197-3/17/09…$15.00
https://doi.org/10.1145/3124680.3124736

*Work in Progress. Pre-print can be accessed at https://arxiv.org/abs/1707.08514

Measure	ext2	ext4	xfs	f2fs	btrfs
File Create					
Write Cost (KB)	24	52	52	16	116
fsync	4	28	4	4	68
checkpoint	20	24	48	12	48
Read Cost (KB)	24	24	32	36	40
Space Cost (KB)	24	52	20	16	116
File Rename					
Write Cost (KB)	12	32	16	20	648
fsync	4	20	4	8	392
checkpoint	8	12	12	12	256
Read Cost (KB)	20	24	48	40	48
Space Cost (KB)	12	32	16	20	392

Table 2: IO Cost for Metadata Operations. ***The table shows the read, write, and space IO costs incurred by different file systems for different metadata operations. The write cost is broken down into IO at the time of*** **`fsync()`*****, and checkpointing IO performed later.***

file deletion, and directory deletion, are similar to our presented results. Table 2 presents the results.

Discussion. IO and space amplification arises in Linux file systems due to using the block interface, from crash-consistency techniques, and the need to maintain and update a large number of data structures on storage. Write amplification is high in our workloads because we do small writes followed by a `fsync()`, which forces file-system activity, such as committing metadata transactions.

With byte-addressable non-volatile memory technologies arriving on the horizon, using such block-oriented file systems will be disastrous. We need to develop lean, efficient file systems where operations such as file renames will result in a few bytes written to storage, not tens to hundreds of kilobytes.

3 THE CREWS CONJECTURE

Inspired by the RUM conjecture [13] from the world of key-value stores, we propose a similar conjecture for file systems: the CReWS conjecture.

The CReWS conjecture states that it is impossible for a general-purpose file system to provide strong crash (C)onsistency guarantees while simultaneously achieving low (R)ead amplification, (W)rite amplification, and (S)pace amplification.

Implications. The CReWs conjecture has useful implications for the design of storage systems. If we seek to reduce write amplification for a specific application such as a key-value store, it is essential to sacrifice one of the above aspects. For example, by specializing the file system to a single application, it is possible to minimize the three amplification measures. For applications seeking to minimize space amplification, the file system design might sacrifice low read amplification or strong consistency guarantees. For non-volatile memory file systems [14, 15], given the limited write cycles of non-volatile memory [16], file systems should be designed to trade space amplification for write amplification; given the high density of non-volatile memory technologies [17–21], this should be acceptable. Thus, given a goal, the CReWS conjecture focuses our attention on possible avenues to achieve it.

REFERENCES

[1] Marshall K McKusick, William N Joy, Samuel J Leffler, and Robert S Fabry. A fast file system for unix. *ACM Transactions on Computer Systems (TOCS)*, 2(3):181–197, 1984.

[2] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33. Citeseer, 2007.

[3] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Alex Tomas Andreas Dilge and, and Laurent Vivier. The New Ext4 filesystem: Current Status and Future Plans. In *Ottawa Linux Symposium (OLS '07)*, Ottawa, Canada, July 2007.

[4] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system. In *USENIX Annual Technical Conference*, volume 15, 1996.

[5] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):9, 2013.

[6] Pradeep J Shetty, Richard P Spillane, Ravikant R Malpani, Binesh Andrews, Justin Seyster, and Erez Zadok. Building workload-independent storage with vt-trees. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 17–30, 2013.

[7] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Wisckey: separating keys from values in ssd-conscious storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 133–148, 2016.

[8] Percona TokuDB. https://www.percona.com/software/mysql-database/percona-tokudb.

[9] Vijay Chidambaram, Tushar Sharma, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Consistency Without Ordering. In *Proceedings of the 10th USENIX Symposium on File and Storage Technologies (FAST '12)*, pages 101–116, San Jose, California, February 2012.

[10] Vijay Chidambaram, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Optimistic Crash Consistency. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Farmington, PA, November 2013.

[11] Thanumalayan Sankaranarayana Pillai, Ramnatthan Alagappan, Lanyue Lu, Vijay Chidambaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Application crash consistency and performance with ccfs. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, pages 181–196, Santa Clara, CA, 2017. USENIX Association.

[12] William Jannen, Jun Yuan, Yang Zhan, Amogh Akshintala, John Esmet, Yizheng Jiao, Ankur Mittal, Prashant Pandey, Phaneendra Reddy, Leif Walsh, et al. Betrfs: A right-optimized write-optimized file system. In *FAST*, pages 301–315, 2015.

[13] Manos Athanassoulis, Michael S Kester, Lukas M Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, and Mark Callaghan. Designing access methods: The rum conjecture. In *International Conference on Extending Database Technology*, pages 461–466, 2016.

[14] Jian Xu and Steven Swanson. NOVA: a log-structured file system for hybrid volatile/non-volatile main memories. In *FAST*, 2016.

[15] Subramanya R Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems*, page 15. ACM, 2014.

[16] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *ACM SIGARCH computer architecture news*, volume 37, pages 14–23. ACM, 2009.

[17] Simone Raoux, Geoffrey W. Burr, Matthew J. Breitwisch, Charles T. Rettner, Y. C. Chen, Robert M. Shelby, Martin Salinga, Daniel Krebs, S. H. Chen, H.L. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465–479, 2008.

[18] Chun Jason Xue, Youtao Zhang, Yiran Chen, Guangyu Sun, J Jianhua Yang, and Hai Li. Emerging non-volatile memories: opportunities and challenges. In *Proceedings of the seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 325–334, 2011.

[19] Yenpo Ho, Garng M Huang, and Peng Li. Nonvolatile Memristor Memory: Device Characteristics and Design Implications. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 485–490. ACM, 2009.

[20] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. The missing memristor found. *Nature*, 2008.

[21] Leon Chua. Resistance switching memories are memristors. *Applied Physics A*, 102(4):765–783, 2011.