Is End-to-End Integrity Verification Really End-to-End?

Ahmed Alhussen
Computer Science and Engineering
University of Nevada, Reno
aalhussen@nevada.unr.edu

Batyr Charyyev
Computer Science and Engineering
University of Nevada, Reno
bcharyyev@nevada.unr.edu

Engin Arslan
Computer Science and Engineering
University of Nevada, Reno
earslan@unr.edu

***Abstract*—End-to-end integrity verification is proposed to improve reliability of file transfers by comparing the checksum of files at source and destination servers. However, state-of-the-art implementation of end-to-end integrity verification fails to detect silent data corruption that may happen during disk writes since checksum computation at the destination server could read cached copy of files from main memory when file size is small. In this work, we show that existing integrity verification methods fail to capture silent disk errors and present an application-level solution to improve the reliability of integrity verification.**

I. Introduction

As data transfer rates are rapidly increasing, traditional integrity verification mechanisms fall short to detect corruption. For example, TCP checksum fails to detect errors once in 16 million to 10 billion packets [1]. As a result, researchers observed up to 5% data corruption for file transfers in 100 Gbps network [2]. End-to-end integrity verification is introduced to avoid silent data corruption by comparing the checksum of files at source and destination servers. However, state-of-the-art integrity verification algorithms could miss disk write errors since checksum computation reads files from cache memory when they are recently written to disk and copy of them are kept in the memory.

Potential solutions to this problem include (i) clearing cache before checksum computation and (ii) bypassing main memory for file reads. However, the first solution could have detrimental impact on system performance and requires root access. On the other hand, bypassing cache memory is discouraged due to poor I/O performance.

II. Proposed Solution

In this work, we propose a Secure Integrity Verification Algorithm (SIVA) that delays checksum computation of a file to avoid reading from the memory. When multiple files are transferred, first transferred files are evicted from cache memory after some time due to limited capacity of the memory. SIVA exploits this cache eviction policy of operating systems and delays the checksum computation of a file to guarantee that its pages are no longer present in the main memory. That is, SIVA will not compute checksum of a file right after its transfer. Instead, it will wait until the file's pages are no longer expected to be present in the memory such that when checksum process attempts to read the file, it has to read it from disk.

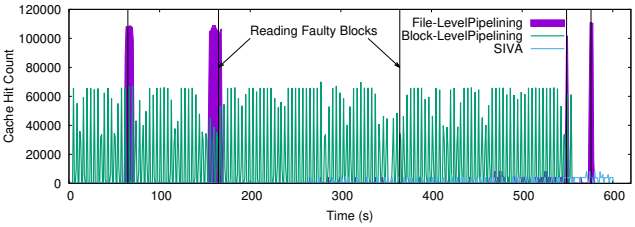

Fig. 1: SIVA keeps cache hits small to detect disk errors.

We compared SIVA against state-of-the-art solutions for end-to-end integrity verification; file-level pipelining and block-level pipelining [3] as shown in Figure 1. File-level pipelining [4] overlaps checksum computation of a file with the transfer of another file to optimize multi-file transfers. Block-level pipelining divides large files into blocks and overlaps transfer of a block with a checksum of another block to improve the benefit of pipelining when files in dataset are in different size. We used a dataset with five files (four files with 1-5 GB size and one with 20GB size). We monitored cache hit values of the algorithms while transfer is running and injected a fault for a randomly chosen block of each file during disk writes operation. We marked the times when file-level pipelining processed corresponding blocks of files. We injected same faults for SIVA and block-level pipelining as well but did not mark in the figure for the sake of simplicity.

We observed that block-level pipelining could not catch any of the faults while file-level pipelining only caught one out of five faults since both operate on memory copy of the blocks. SIVA, however, captured all the faults since its cache hit values are always very small due to reading files from the disk. On the other hand, delaying checksum computation of files causes SIVA to take longer than other algorithms. Hence, SIVA leads to 11% increase in checksum computation time compared to block-level pipelining in exchange of increasing reliability of integrity verification.

III. Conclusion

In this work, we showed that state-of-the-art integrity verification solutions fail to capture silent disk write errors and proposed application-level solution, SIVA, to improve reliability. Preliminary results show that SIVA can capture disk write errors in return of small increase in checksum computation time.

REFERENCES

[1] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," in *ACM SIGCOMM computer communication review*, vol. 30, no. 4. ACM, 2000, pp. 309–319.

[2] R. Kettimuthu, Z. Liu, D. Wheeler, I. Foster, K. Heitmann, and F. Cappello, "Transferring a Petabyte in a Day," *Future Generation Computer Systems*, 2018.

[3] S. Liu, E.-S. Jung, R. Kettimuthu, X.-H. Sun, and M. Papka, "Towards optimizing large-scale data transfers with end-to-end integrity verification," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3002–3007.

[4] "Globus," https://www.globus.org/.