FastVer: Making Data Integrity a Commodity

SIGMOD '21, June 20–25, 2021, Virtual Event, China

Bo Yong Lin
NTU BEBI - First grade
Using AI tool perfect my writing

1 PROBLEM

The problem addressed by the author's team in this paper is the need for a high-performance key-value store to ensure data integrity in large databases.

Ensuring data integrity is a critical challenge in modern distributed systems, especially in cloud-based environments where data is often stored across multiple nodes and managed by untrusted third parties. It is even more important for customers to maintain sensitive information themselves. For example, consider a cloud service that uses passwords to authenticate users. If a rogue administrator can manipulate this table and change passwords undetected, they can log in to the service as another user.

The basic idea of data integrity is to store the cryptographic hash of the data in a secure and trusted location. The basic idea of data integrity is to store a cryptographic hash (such as SHA-256) of the data in a secure and trusted location. Each time it is updated or accessed, it is authorized and compared against this trusted location.

Traditional approaches to ensuring data integrity, such as Merkle trees and memory verification, have limitations that make them unsuitable for high-performance key-value stores.

In particular, Merkle trees are efficient for verifying the integrity of single operations but can be slow when used to verify a large batch of operations. On the other hand, deferred memory verification is faster for verifying batches of operations, but requires scanning the entire database during verification, which is costly.

The author team also defines various performance requirements to describe and compare different solution approaches. It is shown that the final solution, FastVer, satisfies all the required conditions, whereas previous methods do not.

P1: Size of the Verifier State: The performance of the verified database system, however, must degrade gracefully with the size of the memory allocated to the verifier. In particular, a solution should not rely on the verifier storing the whole database to achieve good performance.

P2: Verification Complexity: Verification complexity refers to the additional computation performed by the host and the verifier to validate an operation result. Verification complexity impacts the overall throughput of the verified database. The verification complexity should be $O(1)$ with a low constant for operations over frequently accessed records.

P3: Verification Latency: One of the techniques for data integrity involves verifying operations in a batch for better performance. Suchbatching introduces latency in communicating a validated result of a client operation.

P4: Concurrency Bottlenecks: Verification should not introduce con-currency bottlenecks beyond those arising from the client workload. In other words, the concurrency bottlenecks of verified databases should be no different from that of a regular system without data integrity.

In general, the novel approach used in FastVer, with a hybrid solution that combines the advantages of both approaches, achieves one to two orders of magnitude higher throughput than traditional approaches based on either Merkle trees or memory verification.

2 PRIOR WORK

Before jumping straight into the topic, the author team has big questions about using a verifier as a proxy for the client to check the validity of the result:

- What is a hash that is maintained inside the verifier?
- How is the hash used to verify that the output is correct?

2.1 MERKLE TREES

Merkle trees have been used for several decades to ensure data integrity. In essence, a Merkle tree is a hierarchical structure of cryptographic hashes that are collision resistant and is constructed using database records. In a common Merkle tree, the leaves represent the database records, including a key k and a value v. Each node in the tree stores a hash and points recursively to its parent nodes.

To implement a verified database using Merkle trees, the verifier stores the root, while the host stores the rest of the Merkle tree. To verify the correctness of the $get(k)$ / $put(k, v)$ operations, the verifier checks that each ancestor hash matches the hash computed by its children. If all these checks pass, the verifier validates (signs) the value v. By exploiting the collision-resistant property of hashing, the author team shows that at least one of the checks will fail if v is not the current value of key k.

However, traditional Merkle trees do not meet the performance goals of $P2$ or $P4$. This is because each operation has a logarithmic verification cost or touches the root, respectively.

2.2 DEFERRED MEMORY VERIFICATION

Deferred memory verification relies on a cryptographic primitive called collision-resistant multiset hashing, a hash function over multisets (bags), and does not require additional data structures such as Merkle trees.

In the verification step, the verifier does not have enough information to immediately check the integrity of the record. Instead, it performs some internal accounting and signs a preliminary validations $v(k, v, t, e)$ indicating that it has recorded the operation as part of a batch$e$. If these checks pass, the verifier signs a batch validation $s_v(e)$.

However, it does not satisfy the P3 bounded verification latency objective. Verification latency is linear in database size, as each record is routed to a verifier thread during the verification scan. The author team notes that this is wasted work for infrequently accessed records.

3 SOLUTION

To provide a trusted location in the cloud, the servers are called *enclaves*, which are a major step forward in implementing security features in the cloud. It is a protected region of the virtual memory space of a process containing code and data with a well-defined interface.

This paper achieves strong integrity guarantees and extremely high throughput and latency guarantees by combining the following state-of-the-art ideas:

Caching within the enclave: Exploiting the state within the enclave to cache verification data. Instead of storing only the root of the Merkle tree, our approach allows caching of arbitrary nodes within the enclave. Such caching reduces the cost of the path from a verified record to the root since only the path from a record to the first cached ancestor needs to be verified.

Multiple minimally interacting verifier (threads): Each verifier thread has its clock, cache, read and write set hashes, and does not communicate with other verifier threads except at the end of the verification epoch to aggregate local set hashes.

Hybridizing Merkle and deferred memory verification: The hybrid approach organizes records into a verification hierarchy based on their access characteristics: The integrity of the *cold* database records has the lowest frequency, and nodes are verified by checking their hash against the parent node as in the Merkle tree approach. This approach is expensive because it leads to a logarithmic chain of hash checks, but the cost is tolerable for cold data.

The integrity of *warm* records and Merkle nodes is batch verified using the deferred memory verification approach. It is protected using multi-set hashing by scanning all records, and the maximum tolerated latency of the application influences the number of records that can be protected.

The *hottest* records and Merkle nodes are cached in the enclave, which has the characteristics of no verification, high efficiency, and small size. As these are stored in untrusted memory, there is no integrity check on this data.

Based on FASTER, FASTVER is implemented with 500 lines of C++ code and uses F★ proof assistant to verify its correctness. It provides the same key-value API as FASTER, plus an additional *verify()* method that detects if an unauthorized attacker has tampered with the database and checks that the results of all read operations are consistent with historical updates.

4 RESULT

Even with enclaves, Merkle trees achieve at best tens of thousands of operations per second. Deferred memory verification involves scanning the entire database in the trusted location when verifying a batch of operations which can take dozens of seconds.

The author team's experiments show that FASTVER can process more than 50 million key-value operations/sec with sub-second (1s) verification latency for a range of database sizes and workloads. However, as database size increases, high throughput is associated with increasingly higher verification delays.

For a database of 128M records and 32 worker threads, the author team achieves a sub-second verification latency with a throughput of 10M ops/sec, which is three orders of magnitude better than Merhle trees. In contrast, the best-reported throughput in Concerto, which is the best-known deferred memory verification system in terms of both throughput and latency, is around 3M ops/sec.

5 CRITIQUE

FastVer is an approach to ensuring data integrity in high-performance key-value stores. The combination of Merkle trees and deferred storage verification can offer advantages in verification complexity and throughput over traditional methods, as well as isolated enclaves to protect sensitive workloads, improving overall database security.

Overall, I think the author team has done a good job in providing a *verify()* method built with the API that detects unauthorized tampering with the database and can be useful and easy to use in ensuring data trustworthiness.

6 POSSIBLE EXTENSION

Given the current challenges faced by Taiwan's medical institutions, the only way to achieve information security and privacy is to adopt *Centralized database management* for Taiwan's medical institutions and technology to stay inside the hospitals. If FASTVER can be integrated to create a decentralized records management system capability using the *Decentralised database management* enabled by blockchain technology, it will reduce the need to manage access between patients and records through other organizations without tampering. This would involve looking at how the verifier interacts with the blockchain

network to ensure the validity of actions that all nodes agree to perform on the database.

CITATION

[1] Arvind Arasu, Badrish Chandramouli, Johannes Gehrke, Esha Ghosh, Donald Kossmann, Jonathan Protzenko, Ravi Ramamurthy, Tahina Ramananandro, Aseem Rastogi, Srinath Setty, Nikhil Swamy, Alexander van Renen, and Min Xu. 2021. FastVer: Making Data Integrity a Commodity. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21). Association for Computing Machinery, New York, NY, USA, 89–101. https://doi.org/10.1145/3448016.3457312