Research Report

TRUSTDBLE: Towards a New Class of DBMSs for Data Sharing

RECENTLY, A NEW CLASS OF SYSTEMS FOR SHARED AND COLLABORATIVE DATA MANAGEMENT HAS GAINED MORE AND MORE TRACTION. IN CONTRAST TO CLASSICAL DATA BASE MANAGEMENT SYSTEMS (DBMS), SYSTEMS FOR SHARED DATA NEED TO PROVIDE ADDITIONAL GUARANTEES TO ENSURE THE INTEGRITY OF DATA AND TRANSACTION EXECUTION. IN THIS PAPER, WE PRESENT TRUSTDBLE, A NEW DBMS THAT EXTENDS THE ACID PROPERTIES (I.E., ATOMICITY, CONSISTENCY, ISOLATION, DURABILITY) USED BY CLASSICAL DBMSS WITH A NEW VERIFIABILITY COMPONENT TO ADDRESS THESE NEW REQUIREMENTS.

Carsten Binnig
Muhammad El-Hindi
Simon Karrer
Benedikt Völker

Motivation
Recently, a new class of systems for shared and collaborative data management has gained more and more traction. Examples of such systems include Veritas (Allen et al., 2019), BlockchainDB (El-Hindi et al., 2019), FalconDB (Peng et al., 2020), and Spitz (Zhang et al., 2020).

Compared to classical database management systems (DBMSs) that are designed to be used by a single party, these systems enable multiple parties to manage a shared database (DB) in a collaborative manner. For example, think of a shared database for medical patient records. It would allow hospitals and doctors to directly share and modify patient data to keep track of diagnoses and treatments a patient received. Clearly, shared DBs provide many opportunities not only in the medical domain (e.g., for large-scale epidemic studies), but also for many other fields where access to a shared DB enables more effective collaboration or new use cases (e.g., in the financial domain or supply chains).

However, unlike classical DBMSs, systems for shared data need to provide additional guarantees to ensure the integrity of data and transaction execution, called verifiability guarantees. The main reason for this is that when manipulating a shared DB in a collaborative way, there is often mutual distrust between the different parties that jointly access the shared DB since they often have different interests (e.g., think of an insurance company and a hospital that use a shared database for medical records). Hence, the goal of the verifiability guarantees is to govern the shared database, i.e., to guarantee that the shared database is only modified based on a predefined and agreed upon set of transactions that every party adheres to and that none of the parties can tamper with the data.

If we now look at how existing systems for shared data (such as those mentioned at the beginning) provide verifiability, we can observe that these systems typically take a very implementation-centric approach and often do not integrate well with the ACID guarantees of classical DBMSs. Also, the concrete verifiability guarantees that existing systems provide vary significantly and are hard-baked into their execution model.

Vision
In this paper, we propose to take a more principled and more database-centric approach to provide verifiability for shared data systems. We present the main concept behind our system, TRUSTDTBLE, which is to extend the ACID properties used by classical DBMSs with a new verifiability component resulting in the ACID-V properties. Similar to the other components in ACID, such as the well-known isolation property, TRUSTDBLE allows to specify the guarantees of verifiability in a declarative manner using different verification levels (i.e., strict or more loose). We believe that extending the ACID properties with verifiability is not only a natural fit and gives applications well-defined guarantees, but it also enables a new class of shared DBMSs that decide based on the verification level what optimizations and concrete execution strategies are required for the desired guarantees.

From ACID to ACID-V
Adding the V to ACID. In classical databases, transactions are governed by the ACID properties. As mentioned before, the concrete properties that should be satisfied can be defined declaratively and are implemented by databases in various ways. For example, for the I(solation) in ACID, a user can declare the specific isolation level (e.g., read committed, serializable) that a transaction should run under. This isolation level is, then, guaranteed by a database through its concurrency control scheme (e.g., optimistic or pessimistic). Similarly, we propose to add a new verifiability property that a user can specify declaratively and that database systems can implement in different ways. Looking at verifiability from a conceptional perspective allows to reason about the guarantees a system provides independent from implementation details.

To add the V to ACID, we extend the classical transaction state model of ACID-compliant DBMSs by a verified state. For simplicity, Figure 1 visualizes the extended state model for ACID-V for the case in which all nodes in a shared DBMS act honestly. We will discuss malicious behavior in follow-up work. In our state model, a transaction can only reach the verified state after it reached the committed state.

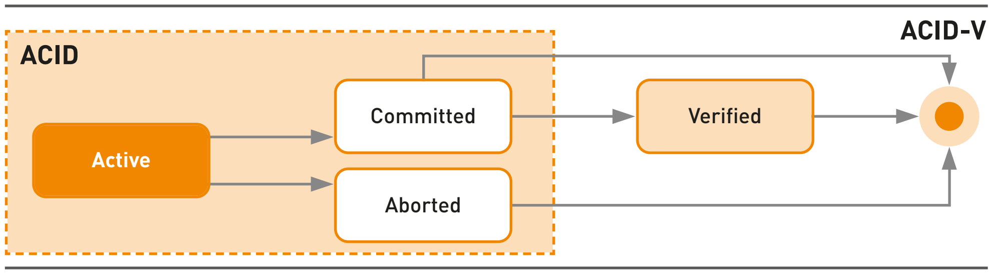

Figure 1: Simplified State Model for ACID-V (the classical transaction state model is extended with a verified state)

Modeling verified as a state that follows the committed state has several advantages. First, since verification is typically an expensive step, the model leaves some freedom when the transition from committed to verified happens (i.e., directly after the commit or deferred). Moreover, it enables the user to declare which state is allowed to be read by other transactions (e.g., if committed, but unverified can be read or if all state must be verified before becoming visible). Second, the verified state is an optional state as shown in Figure 1, i.e., not all committed transactions need to be verified, which allows partial verification to reduce the overhead introduced by verification.

Verification Levels. While a formal definition of ACID-V and a more complete discussion of possible verification levels are out of scope for this paper, in the following, we show how a first set of different verification levels can be defined based on the state model we introduced before. Based on this, we will discuss what implications different verification levels can have on the integrity of data/execution and a system's performance.

Strict Verification (SV): This verification level requires that all transactions need to be verified. Moreover, all transactions are allowed to read only verified state. A similar guarantee can be provided by the online verification schemes of existing systems such as Veritas and BlockchainDB which guarantee that the result of a transaction is verified before becoming visible to other transactions. In terms of transaction execution, this level implies that transactions should transition from the committed state to verified as fast as possible. Otherwise, this can lead to low performance or in worst case starvation (i.e., if there are too many unverified transactions). Clearly, strict verification, thus, has a high overhead and might lead to inferior performance when compared to more relaxed levels that we discuss below.

Unstrict Verification / full (UV-f): Compared to the previous level, UV-f is a more relaxed verification level since it allows transactions to read from committed but not yet verified state. That is, even if the verification of a transaction is still pending, other transactions can access its committed state. However, all transactions are still being verified (hence, it is called full) and unsuccessful verification in case of malicious behavior needs to be handled. In contrast to the SV level, though, this makes room for different optimizations. Most importantly, transactions are not blocked by potentially expensive verification protocols since verification can be executed in batches and in a deferred manner. This is similar to deferred verification schemes available in existing systems.

Unstrict Verification / partial (UV-p): This verification level relaxes the guarantees of the previous level (UV-f) even further. Similar to UV-f, transactions are allowed to access committed, but unverified state. Further, in partial unstrict verification (UV-p), we do not enforce that all transactions need to be verified. Consequently, this verification level assumes that verified is an optional state of a transaction. In this level, a user can, thus, explicitly request to verify only a subset of transactions. Hence, UV-p could be used to limit the verification overhead to some (e.g., important) transactions or to provide probabilistic guarantees by verifying only a sample of all transactions.

Conclusion and Future Directions

In this paper, we presented our vision for TRUSTDBLE as an ACID-V-compliant DBMS for data sharing. As a core contribution, we propose to specify the guarantees of verifiability in a declarative manner and let the DBMS decide what optimizations and concrete execution strategies are best suited to meet the guarantees of a particular verification level. We think that the model of ACID-V-compliant DBMSs can trigger many follow-up work. For example, the verification levels proposed in this paper are just an initial direction and a more profound discussion of what levels data sharing applications actually need is required. Further, similar to isolation levels that have triggered different implementation strategies, we think ACID-V will also enable a wide variety of different implementation strategies to achieve the desired guarantees of verification.

References

Allen, L.; Antonopoulos, P.; Arasu, A.; Gehrke, J.; Hammer, J.; et al.:
Veritas: Shared Verifiable Databases and Tables in the Cloud.
In: 9th Biennial Conference on Innovative Data Systems Research (CIDR); Asilomar (CA), U.S., 2019.

El-Hindi, M.; Binnig, C.; Arasu, A.; Kossmann, D.; Ramamurthy, R.:
BlockchainDB: A Shared Database on Blockchains.
In: Proceedings of the VLDB Endowment; Los Angeles (CA), U.S., 2019; pp. 1597–1609.

Peng, Y.; Du, M.; Li, F.; Cheng, R.; Song, D.:
FalconDB: Blockchain-Based Collaborative Database.
In: Proceedings of the 2020 ACM SIGMOD; Portland (OR), U.S., 2020; pp. 637–652.

Zhang, M.; Xie, Z.; Yue, C.; Zhong, Z.:
Spitz: A Verifiable Database System.
In: Proceedings of the VLDB Endowment; Tokyo, Japan, 2020; pp. 3449–3460.