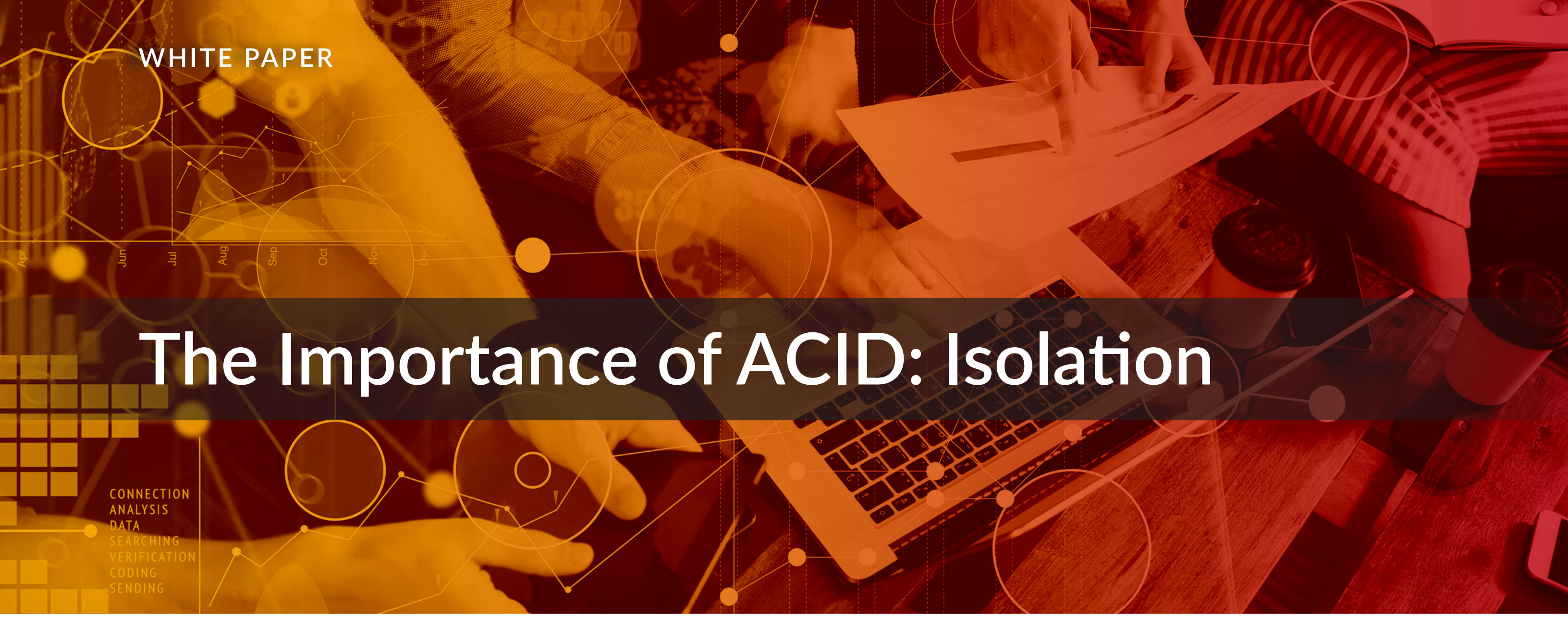

WHITE PAPER

The Importance of ACID: Isolation

Introduction

With the explosion in both the number and types of data management systems over the last few years, there has also come confusion about the definitions of terms that had very specific and deliberate meanings in the world of relational databases. We often hear that a new product, surprisingly sometimes even a NoSQL system, claims to be ACID-compliant, but what does that really mean?

It turns out that the ANSI definitions for ACID (Atomicity, Consistency, Isolation, and Durability) include different levels of compliance. Two systems can both accurately claim to be ACID-compliant, but if they support different levels of the four components of ACID, in practice they will act very differently. The differences become particularly apparent and important under heavy workload and during failures, especially network connectivity failures between nodes of the same cluster. Most databases that claim to be 100% ACID compliant use very vague or misleading definitions. This applies to large established vendors as much as to small new startups — so understanding what is being claimed, and more importantly what you actually get in practice, is important. That makes services like Kyle Kingsbury's Jepsen testing critically important (http://jepsen.io).

Isolation

The point of a database is to maintain a steady, consistent, reliable set of data that can be used by applications. If there was one user accessing data on one machine, that would be a relatively easy task since there would be only one transaction occurring at any one time. But data management systems have evolved to support web-scale applications and now most modern databases, whether SQL, NoSQL or NewSQL, support thousands of concurrent users across clusters of servers.

Fundamental to achieving the concurrency control that is needed in every data management solution is isolation. The highest level of isolation ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. With effective isolation, an incomplete transaction cannot affect another incomplete transaction. Isolation is probably the most important of the ACID properties for ensuring a stable, consistent database, yet it is the property most likely to be relaxed by default due to its negative impact on performance.

Transactions are concurrency control mechanisms, and they deliver consistency even when being interleaved. Isolation brings us the benefit of hiding uncommitted state changes from the outside world, as failing transactions shouldn't ever corrupt the state of the system. Isolation is achieved through concurrency control using pessimistic or optimistic locking mechanisms.

> *There have been some excellent and detailed papers and articles written on the topic. Peter Bailis is a recognized leader in this area — see his paper called "Understanding Weak Isolation Is a Serious Problem" at* http://www.bailis.org/blog/understanding-weak-isolation-is-a-serious-problem/

Isolation Levels and Phenomena

The mechanism for isolation is purposely left open in the ANSI definitions. Although some database management systems offer MVCC, usually concurrency control is achieved through locking. Locking, however, increases the serializable portion of the executed code, affecting parallelization and thus, scalability.

The ANSI/ISO SQL specifications define four isolation levels: (1) READ UNCOMMITTED, (2) READ COMMITTED, (3) REPEATABLE READ, and (4) SERIALIZABLE.

Along with these levels, the specification also defines three phenomena which may or may not be consequences of the various isolation levels. Those phenomena are:

Dirty read
A dirty read happens when a transaction is allowed to read uncommitted changes of some other running transaction. This happens because there is no locking preventing it.

Non-repeatable read
A non-repeatable read manifests when consecutive reads yield different results due to a concurring transaction that has just updated the record we're reading. This is undesirable since we end up using stale data. This is prevented by holding a shared lock (read lock) on the read record for the whole duration of the current transaction.

Phantom read
A phantom read happens when a second transaction inserts a row that matches a previous select criteria of the first transaction. We therefore end up using stale data, which might affect our business operations. This is prevented using range locks or predicate locking.

The relationship between the isolation level and the possible phenomena are shown below:

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Not Possible	Possible	Possible
REPEATABLE READ	Not Possible	Not Possible	Possible
SERIALIZABLE	Not Possible	Not Possible	Not Possible

Why is Isolation needed for FinServ?

If there is one industry for which immediate and accurate answers are important it would be FinServ. Money is the obvious example — online applications are manipulating money in the classical transactional model. You buy something, using your credit card, and many "transactions" that need to occur for you to get your purchased goods delivered. At the same time you are online shopping, hundreds or thousands of other people are as well, perhaps some of whom are looking at the same merchandise you are. They may be eyeing that same Nintendo Switch that happens to be the last one in inventory at your local Walmart and even putting into their shopping cart. What happens when 100 people put the same one remaining item into their shopping cart? And maybe at the same time, your spouse is using that same credit card in a physical store somewhere, creating a contention for your purchasing limit. Isolation of transactions becomes a critical component to online applications being able to deal with contentions like this, particularly for finite and limited resources like inventory, time, and money.

Here are some sample use cases we have seen where VoltDB's strong isolation makes it faster and easier to create accurate, reliable applications.

SLA/Regulatory Management

If there is one industry for which immediate and accurate answers are important it would be FinServ. Money is the obvious example — online applications are manipulating money in the classic transactional model. You buy something using your credit card, and many "transactions" need to occur for you to get your purchase. Tracking the status of transactions is also important in the highly-regulated FinServ industry. Many companies come to VoltDB looking for solutions that will provide them with accurate, auditable data about data processing for trades and other controlled activities. Eventual consistency doesn't cut it for these applications — at any given point in time they need a definite, accurate answer about how much time a given process took, an amount of time that is often highly regulated, to show that the company complied with regulations — or they are liable for punitive fines. Accurate accounting systems are worth a significant amount to these companies to avoid fines and uncertainties that could potential result in fines.

Single Source of Truth (SSoT)

We hear from more FinServ customers who are looking to add a data management platform to front-end multiple Systems of Record to create a "single source of truth" for their regular consumers of data. Those SoRs are overwhelmed by daily lookup requests, with some writes thrown in, so these customers want to add a caching layer that can take the brunt of ad-hoc queries throughout the day, buffering content and reducing the load on the SoRs. If limited to reads, many products could be used to present this unified view to corporate business intelligence users and satisfy casual ad-hoc queries. But when you want to have a caching layer take writes to correct or update records and you want that new information to be guaranteed to flow back to the systems of record, you need to have strong ACID, particularly isolation, to make sure the records are updated consistently and logically based on the requests. In-memory databases with strong ACID compliance are perfect solutions for such a use case.

Precise Real-time Resource Management — “The Last Dollar Problem”

Many of our customers offer applications that manage different kinds of budgets. Whether it is spending up or down on an investment account in FinServ, spending down an ad budget in AdTech, or tracking mobile phone usage against plan in Telco, accurate and current accounting of the budget amount is critically important so you know what you can work with at that moment in time. If you can’t accurately assess what the balance is, you need to resort to building in a buffer which means that you either risk overspending based on the amount in the account, or you underspend, leaving money (or minutes) on the table. Strict isolation means you have a more accurate view of exact amounts that are allocated and used, giving you an immediate, more accurate picture of how much you have left to use.

Exactly Once Semantics

A common use case for web applications is making sure something happens once and only once for a given event. This is sometimes associated with sessionization, where there are a bunch of events arriving at different times, sometimes out of order, that need to be associated with one particular activity. This can be one particular call on a cellular network or one ad auction or one financial transaction that spans online and in-store.

Providing at least once semantics is easy — you can repeat an event multiple times and rely on the receiving logic to deduplicate it. But exactly-once is much more complicated — it typically involves setting a unique id for each activity and associating events with that activity. That means being able to get agreement from all the members of the cluster on that unique id (really a counter) and matching up each event to those ids. Strict isolation helps enable accurate accounting of the already established ids and the matching of incoming events to those ids.

Conclusion

VoltDB promises serializable ACID consistency, the strongest isolation guarantee available, and we’ve been verified by Jepsen testing to that level. We do this even when distributed across a cluster. Our automated testing proves every feature works even when faced with every kind of failure. Our testing, including ongoing Jepsen testing, ensures we continue to meet our claimed ACID compliance and our goal of making the fastest, most transactionally-powerful and safest system available.

20July2017