Operation Ordering in Systems

Consistency models help programmers reason about the behavior of complex systems by providing a guarantee regarding the order in which the system will execute operations. Consistency models come in different levels based on the strength of their ordering guarantees: strong consistency models (e.g., linearizability) impose more limitations on execution ordering than weak consistency models (e.g., eventual consistency). This difference leads to a classic trade-off between consistency models and performance.

Unfortunately, researchers in systems, architecture and databases have developed different and confusing terminology for models that limit execution ordering. This article clarifies the difference between consistency, coherence and isolation, as well as some popular models for each, including, sequential consistency, serializability and linearizability.

Consistency vs. Coherence vs. Isolation

Before we dive into the different levels of consistency, coherence and isolation, I will define the guarantees themselves.

Consistency, as in the CAP theorem, defines how *copies of the set of data items* in a system relate to each other. Does the system ensure that they appear as if they are a single copy (to an application or an external observer)? Or do they appear to have some other apparent behavior (i.e., after some period of time they converge to a single copy)?

Coherence defines how *copies of a single item* relate to each other (assuming that the system holds more than one data item). Architects tend to define this differently from consistency due to reasoning about a single cache line vs. multiple cache lines. As a result, it is possible to have consistency bugs that are not coherence bugs.

Isolation, in ACID, defines the how *sets of multiple operations on the set of data items* in a system relate to each other. Does the system ensure that a set of operations (grouped together in a *transaction*) appear as if they ran at a single point in time? Do transactions then appear as if they each ran sequentially on a single copy of the database? Or some other behavior (i.e., each transaction ran on its own snapshot of the database)?

In summary, (1) coherence dictates how a system orders operations for a single data item in the system, (2) consistency dictates how a system orders operations for the entire set of data items in the system, and (3) isolation dictates how a system orders sets of operations grouped in transactions over the entire set of items in the system. As a side effect, consistency is a total ordering guarantee for a set of replicas, while isolation is a partial ordering guarantee (because not every transaction touches every data item).

What about the C in ACID?

Unlike isolation and CAP consistency, the ACID type of consistency is not a system guarantee, but an application-specific invariant (e.g., bank account balance must be always greater than 0). Enforcing this type of consistency depends on the application as well as the system. For example, the A, I & D system guarantees ensure that application doesn't see uncommitted or other intermediate state, but the application needs to ensure that it does not violate its own invariants (e.g., the application needs to check the bank account balance before withdrawing); otherwise, the database will abort the transaction.

I've been reading some great old database papers recommended by Phil Bernstein, and Thomas [1] defines, *internal consistency* as the C in ACID and *mutual consistency* as the C in CAP (both of which should not be confused with external consistency!), which I think is a great way to differentiate the two.

As Thomas says,

> The notion of internal consistency is somewhat more difficult to define precisely. It concerns the preservation of invariant relations that exist among items within a database. As such, internal consistency is related to the interpretation or semantics of items in the database. Therefore, most of the responsibility for the internal consistency of a database must rest with the application processes which update it.

Support for this kind of consistency is available in modern database systems, but rarely discussed. As a result, when people talk about transactional consistency or database consistency it becomes confusing, they almost always mean isolation.

Linearizability, Serializability and Others

OK, now back to how we order operations and transactions in a storage system. There are so many different levels of ordering guarantees that this is a huge topic. I will primarily talk about the strongest variants because these are the best defined.

Sequential (or serializable) consistency ensures that the same operations are applied in the same order to every copy of a set of data items.

Serializable isolation ensures that transactions execute as if they were executing one at a time on a single copy of the database.

Linearizable consistency ensures that the same operations are applied in the same order to every copy of the data item and that the order reflects the order in which the operations *appear* to execute to an external observer (like the application).

Strict serializable isolation ensures that transactions execute as if they were executing one at a time on a single copy of the database and that the order matches the order in which the transactions *appear* to execute to an external observer (like the application).

There has been much discussion of externally consistent (or linearizable) transactions since the publication of Google's Spanner system [2]. Traditionally, the guarantees provided by Spanner and applied to transactions would be called strict serializability. Practically, it doesn't matter: *linearizability is just strict serializability for single operations.*

Strong isolation without strong consistency

It might seem that consistency and isolation are tied together in a database system. If a database has one level of isolation (e.g., strict serializability) for transactions, then it typically replicates single data items with a matching level of consistency (e.g., linearizability). Or put another way, if you want a strict serial ordering of transactions, you need to start with a strict serial ordering of operations, right? (See why this is confusing?)

Interestingly, we demonstrated that this is not necessary in the **TAPIR project (//irenezhang.net/research/tapir/)**[3]. TAPIR is a new protocol that provides strong isolation guarantees without any consistency guarantees. Our observation is that there is no need to enforce both isolation *and* consistency in a transactional storage system that uses replication. This has all kinds of benefits, like letting us get away without using an expensive replication protocol like Paxos.

Summary

- The C in CAP and the C in ACID are not the same thing.
- When people talk about strong consistency in databases, they typically mean strong isolation.
- It is possible to have strong isolation (or what people call strong database consistency) without strong consistency (as demonstrated in TAPIR).

References

[1] R. H. Thomas. **A majority consensus approach to concurrency control for multiple copy databases (http://dl.acm.org/citation.cfm?id=320076)**. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

[2] J. C. Corbett et al. **Spanner: Google's globally-distributed database (http://research.google.com/archive/spanner.html)**. In *Proceedings of OSDI*, 2012.

[3] I. Zhang et al. **Building Consistent Transactions with Inconsistent Replication (/papers/tapir-sosp15.pdf)**. In *Proceedings of SOSP*, 2015.

CONTACT

irene.zhang @microsoft.com

Office 2869, Building 99

14820 NE 36th St. Redmond, WA

iyzhang@github (https://github.com/iyzhang/)

schemeprincess@twitter (https://twitter.com/schemeprincess)