the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Distributed Consistency and Session Anomalies

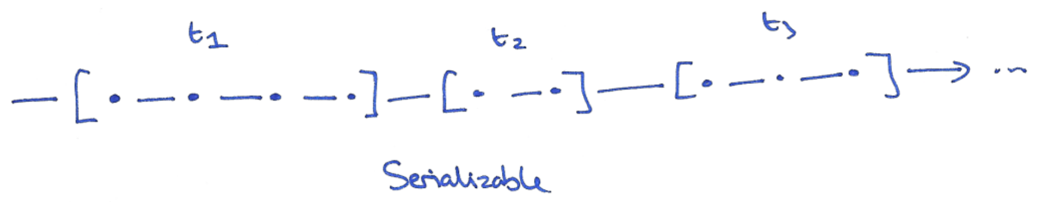
FEBRUARY 26, 2016

tags: Consistency, Distributed Systems, Transaction processing

Since we've spent the last couple of days sketching anomaly diagrams and looking at isolation levels, I wanted to finish the week off with a quick recap of *session anomalies* and *consistency levels* for distributed stores. In terms of papers, I've drawn primary material for this from:

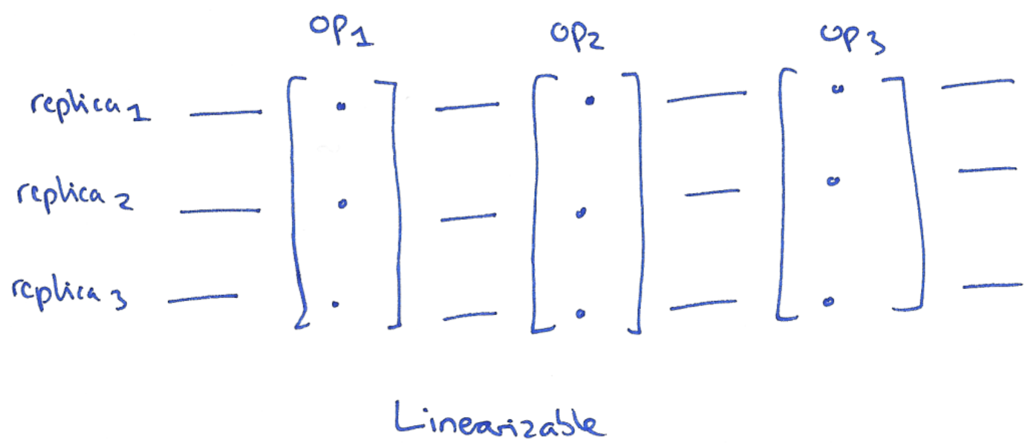
> **Highly Available Transactions: Virtues and Limitations (https://blog.acolyer.org/2014/11/07/highly-available-transactions-virtues-and-limitations/)**, and
> **Linearizability vs Serializability (http://www.bailis.org/blog/linearizability-versus-serializability/)**

In the database systems community, the gold standard is *serializability*. We've spent plenty of time looking at this in the last couple of days. Serializability concerns transactions that group multiple operations across potentially multiple objects. A serializable schedule is one that corresponds to some ordering of the transactions such that they happen one after the other in time (no concurrent / overlapping transactions). It's the highest form of *isolation* between transactions.



(https://adriancolyer.files.wordpress.com/2016/02/serializable.png)

In the distributed systems community, the gold standard is *linearizability*. Linearizability concerns single operations on single objects. A linearizable schedule is one where each operation appears to happen atomically at a single point in time. Once a write completes, all later reads (wall-clock time) should see the value of that write or the value of a later write. In a distributed context, we may have multiple replicas of an object's state, and in a linearizable schedule it is as if they were all updated at once at a single point in time.



(https://adriancolyer.files.wordpress.com/2016/02/linearizable.png)

Linearizability doesn't just apply in a distributed context, although that's what I'm focusing on here. We can also talk about linearizability of operations in a multi-processor system for example.

Strict serializability btw. is serializability & linearizability combined – the transactions are ordered according to real time. It's what you'd get if your database had only a single thread and processed transactions one-at-a-time in order of submission on that thread.

When we have distributed datastores, all the terminology can get a bit confusing because the two communities use the same words in different ways – especially the term 'consistency'.

What do we talk about when we talk about consistency?

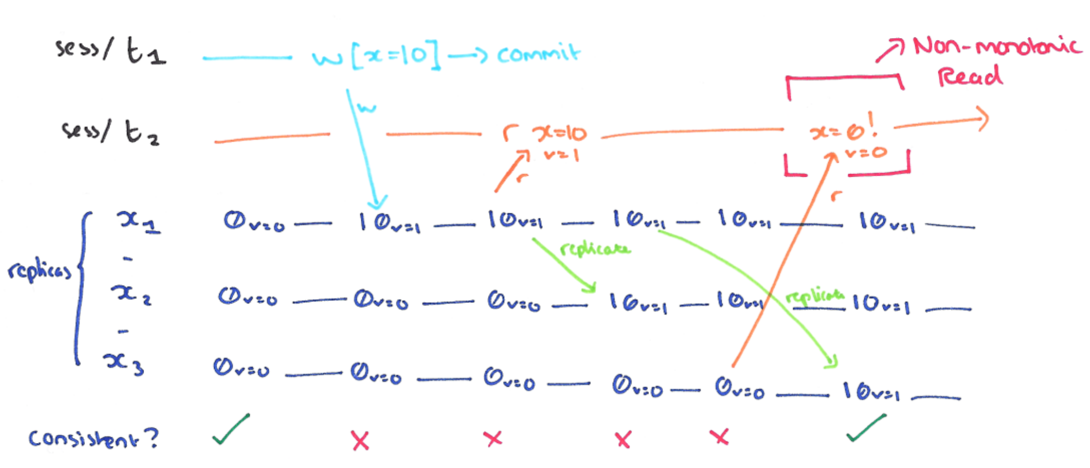
The kind of consistency we've been talking about in the last couple of days is the consistency of database state that arises when all of the data integrity constraints are met. It is the C in ACID. Transactions move the database from one consistent state to another, and because they are atomic and (when serializable) fully isolated, you never observe inconsistent intermediate states.

The kind of consistency we talk about when we talk about 'eventual consistency' or 'causal consistency' is different – here we are concerned with whether or not the state is consistent across multiple distributed replicas (i.e. the same in each of those replicas). It is the C in the CAP theorem. For this reason linearizability is also sometimes called *atomic consistency* – it gives the illusion that consistency is instant and all-or-nothing on writes. This distributed systems version of the term consistency is related to **consenus (https://blog.acolyer.org/2015/03/01/cant-we-all-just-agree/)**. You'll also come across the term *convergence* which is the process by which different replicas become consistent (converge on the same state) over time.

Session Anomalies

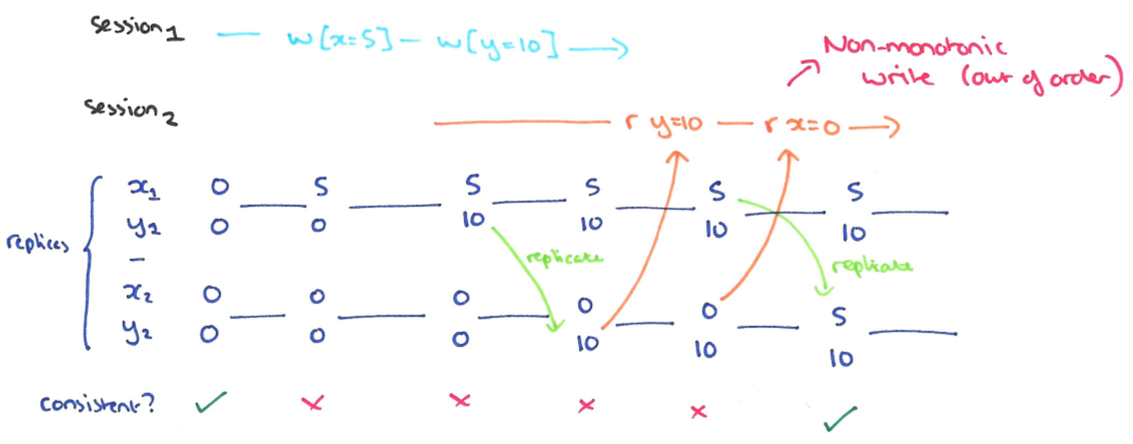
Both serializability and linearizability require (potentially expensive) coordination. Therefore just as we consider weaker isolation levels than serializable, we can also consider weaker consistency levels than linearizable. At weaker consistency levels non-linearizable phenomena (anomalies) can be observed.

A *non-monotonic read* anomaly occurs within a session when a read of some object returns a given version, and a subsequent read of the same object returns an earlier version. The *Monotonic Reads* guarantee prevents these anomalies and ensures that reads from each item progress according to a total order.



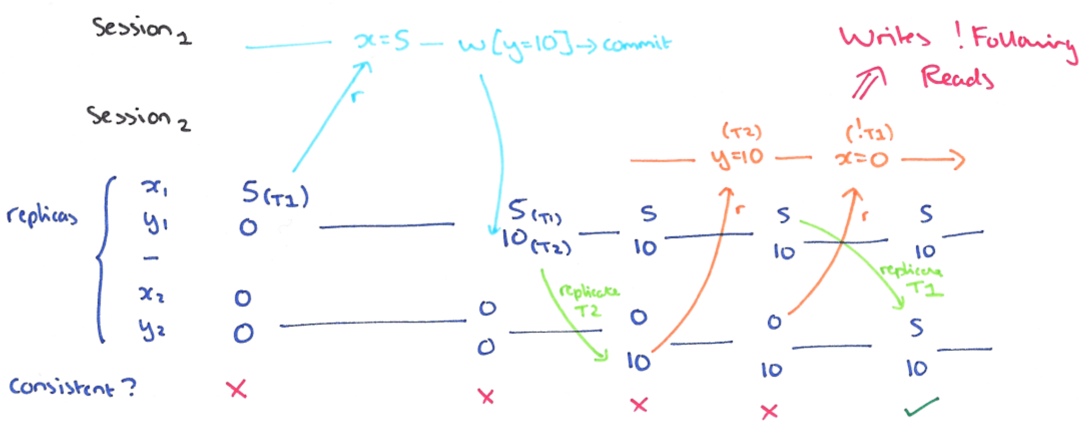
(https://adriancolyer.files.wordpress.com/2016/02/monotonic-reads.png)

A *non-monotonic write* anomaly occurs if a session's writes become visible out of order. The *Monotonic Writes* guarantee prevents these anomalies and ensures that all writes within a session are made visible in order.



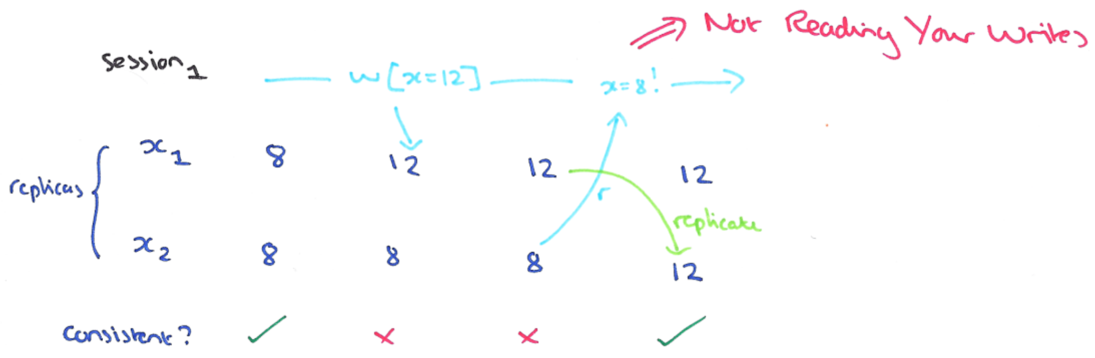
(https://adriancolyer.files.wordpress.com/2016/02/monotonic-writes.png)

A *non-monotonic transaction* anomaly occurs if a session observes the effect of transaction T1 and then commits T2, and another session sees the effects of T2 but not T1. The *Writes Follow Reads* guarantee prevents these anomalies.



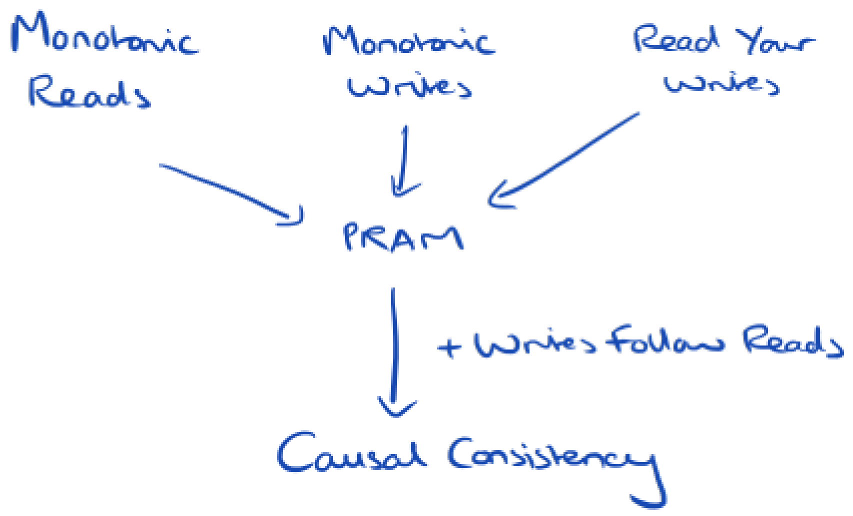
(https://adriancolyer.files.wordpress.com/2016/02/writes-follow-reads.png)

The *Read Your Writes* guarantee ensures that whenever a client reads a value after updating it, it sees that value (or one installed after it). As far as I know, the anomaly that arises when you do not read your writes does not have a name. Perhaps we could call the failure to read your writes an arrest anomaly ;).



(https://adriancolyer.files.wordpress.com/2016/02/read-your-writes.png)

If we combine the Monotonic Reads, Monotonic Writes, and Read Your Writes guarantees we get PRAM (Pipelined Random Access Memory) consistency. If we add Writes Follow Reads to PRAM, we get Causal Consistency.



(https://adriancolyer.files.wordpress.com/2016/03/causal-consistency.png)

from → Uncategorized

12 Comments leave one →

1. **FPJ PERMALINK**
   February 26, 2016 11:22 am
   Very nice post, thank you. Interestingly, it looks like a PRAM shared memory system as described in the TR of Lipton and Sandberg [1] does guarantee read your own writes. A read reads from the local memory, a write writes locally before starting a global write, and a write does not wait for the global write to finish. The Tanenbaum book and other references do refer to them interchangeably, though. But, even the Tanenbaum definition leaves the behavior of the writer open: Writes done by a single process are seen by all other processes in the order they were issued. It does not say anything about the behavior of the writer.

   [1] ftp://ftp.cs.princeton.edu/techreports/1988/180.pdf

   REPLY
2. **Marc Shapiro PERMALINK**
   February 26, 2016 12:15 pm
   It should be emphasised that Serialisability is *non-monotonic*. Each transaction is considered independent; if a client performs transaction T1 followed by T2, and T2 does not read the result of T1, serialisability allows T2 to commit before T1. (This can't happen with Strict Serialisability or with Snapshot Isolation.)

   Also, small correction: the "C" in ACID is the *assumption* that the application program is correct (each transaction in isolation brings the database from a correct state to another correct state), whereas the AID properties are *guarantees* provided by the database. Together they imply serialisability.

   REPLY
3. **Felix PERMALINK**
   March 13, 2016 1:33 pm
   Hey, good article. Seems though you got the last part slightly wrong:
   PRAM = Monotonic Reads + Montonic Writes + Read Your Writes (!)
   Causal Comsistency = PRAM + Writes follow Reads

See the Bailis' HAT paper as a reference.

REPLY

- **adriancolyer PERMALINK***
  March 13, 2016 7:40 pm
  Fixed! Many thanks for the catch 🙂

  REPLY

  - **Felix Gessert PERMALINK**
    April 27, 2016 1:33 pm
    Is it okay, if we use your very nice illustrations for teaching?

  - **adriancolyer PERMALINK***
    April 27, 2016 2:50 pm
    Yes of course, if they can be helpful then please do!

4. **Tuxdude PERMALINK**
   March 17, 2016 7:12 am
   Really awesome explanation Adrian about the Consistency Levels in a Distributed System. I think you missed explaining about Serializability. Some info on that and how it is stronger than causal would be useful.

   It would be also useful to know about the weaker forms of Serializability in Databases Terminoligies like Repeatable Reads, Snapshot Isolation, Cursor Stability

   REPLY

   - **adriancolyer PERMALINK***
     March 17, 2016 12:24 pm
     Thank you! For the other topics you mention, see the posts from the preceding two days: https://blog.acolyer.org/2016/02/25/generalized-isolation-level-definitions/, and https://blog.acolyer.org/2016/02/24/a-critique-of-ansi-sql-isolation-levels/.

     Regards, Adrian.

     REPLY

   - **Marc Shapiro PERMALINK**
     March 17, 2016 12:54 pm
     Serialisability is not stronger than SI or causal consistency. They are mutually incomparable. Serialisability allows the folowing anomaly w.r.t. Causality: user performs transaction T1 followed by T2; database executes them in the order T2 followed by T1. SI allows the write-skew anomaly, which SER disallows.

     The Strict Serialisability model is the only one stronger than both SI and CC (and Serialisability for that matter).

     REPLY

     - **Tuxdude PERMALINK**
       March 17, 2016 4:40 pm
       I'm sorry. I wanted to type Sequential Consistrncy, not Serializability. I agree about Serializability not being stronger than Causal.

The way I understood was Sequential was in between Linearizeable and Causal. It preserves the relative order of all operations within a process, not just causally related ones. But it does not have a global timestamp, so only has a total order amongst all the operations.

Trackbacks

1. The SNOW theorem and latency-optimal read-only transactions | the morning paper
2. The many faces of consistency | the morning paper