2019-06-18 12:32 UTC

Fun MySQL fact of the day: everything is two-phase

Yesterday, we started thinking about InnoDB's **redo log**, and I left you with a thought to consider: how can MySQL keep multiple storage engines' **redo log**s in sync with the binary log. Today, we'll consider how MySQL does it.

You may have heard the term, "two-phase commit" (or 2PC) before, but if not, it is an algorithm used to coordinate multiple systems participating in a distributed, or global, transaction. The most common standard for two-phase commits is the X/Open XA specification, often just called "XA" (for eXtended Architecture). Today, we will consider the XA standard only at the highest level, but if you're the kind of person that likes to maximise fun, you can read the full technical standard. If you *have* heard the term "two-phase commit" before, you probably heard it used with distaste. Maybe something along the lines of "two-phase commit is [bad]" or "two-phase commit doesn't scale". And, hey, while I'm not going to argue too strongly about that, you may be surprised to learn that MySQL uses XA internally for all transactions to ensure that all storage engines participating in a transaction maintain the properties of ACID. Let's see how.

When MySQL gets a request to `COMMIT`, it acts as a coordinator for the transaction (a "transaction coordinator"). The first step is to call `MYSQL_BIN_LOG::prepare`, which tells each storage engine to commit resources to the transaction and ensure that the transaction will succeed. In InnoDB, this call is eventually resolved to `innobase_xa_prepare` wherein InnoDB ensures that the transaction is fully recorded in the **redo log** and syncs the **redo log** file to disk (as necessary). At this point, InnoDB considers the transaction "prepared", but not yet committed. Should MySQL crash *now*, InnoDB would roll-back the in-flight transaction on recovery.

Then, once MySQL has received acknowledgement from all storage engines involved in the transaction (i.e. they've all returned a "yes" vote), `MYSQL_BIN_LOG::commit` will be called and the transaction will be recorded to the binary log. Next, the storage engines will be instructed to commit their changes. In InnoDB, this call is eventually resolved to `innobase_commit` wherein InnoDB lazily logs that the previously-prepared transaction is now "committed". At this point, the transaction is effectively committed and present in both the binary log and the InnoDB **redo log**.

So, you may be wondering: what happens if MySQL crashes after writing its own binary log events but before calling `innobase_commit`? Well, in this case, when MySQL starts back up, in checks whether or not its binary log was safely closed. In this case, it wouldn't have been, so MySQL will collect a list of XA transaction ids (`Xid`s) from the binary log and tell each storage engine to commit them---if they exist and have not *yet* been committed. Remember, this is safe because, by this time, all storage engines have already recorded the transaction before voting "yes" during the prepare phase.

So, next time somebody tells you, in absolute terms, that XA is "bad" or "doesn't scale", you can share this fun fact with them and see how they change their mind.

acid, binlog, databases, innodb, mysql