How do two-phase commits prevent last-second failure?

Asked 16 years, 2 months ago Modified 5 years, 6 months ago

Viewed 14k times

I am studying how two-phase commit works across a distributed transaction. It is my understanding that in the last part of the phase the transaction coordinator asks each node whether it is ready to commit. If everyone agreed, then it tells them to go ahead and commit.

What prevents the following failure?

1. All nodes respond that they are ready to commit
2. The transaction coordinator tells them to "go ahead and commit" but one of the nodes crashes before receiving this message
3. All other nodes commit successfully, but now the distributed transaction is corrupt
4. It is my understanding that when the crashed node comes back, its transaction will have been rolled back (since it never got the commit message)

I am assuming each node is running a normal database that doesn't know anything about distributed transactions. What did I miss?

database distributed-transactions

Share

Improve this question

Follow

edited Oct 28, 2008 at 18:48

community wiki
4 revs, 3 users 68%
Gili

2 Your assumption of a normal database is incorrect. Any resource (message queue, database, etc.) supports distributed transactions (by supporting a transaction coordinator) or it doesn't. If it doesn't, kludges to include it in a distributed transaction will compromise reliability. – erickson Jan 14, 2009 at 19:05

Erickson, if you fold in other people's answers with your comment I will mark it as the official answer. – Gili Jan 14, 2009 at 19:15

5 Answers

Sorted by: Highest score (default)

57

No, they are not instructed to roll back because in the original poster's scenario, some of the nodes have already committed. What happens is when the crashed node becomes available, the transaction coordinator tells it to commit again.

Because the node responded positively in the "prepare" phase, it is required to be able to "commit", even when it

comes back from a crash.

Share Improve this answer

Follow

answered Oct 5, 2008 at 12:23

Jason Kresowaty

16.5k ● 9 ● 60 ● 85

I don't understand how the crashed node is supposed to be able to replay the transaction up to the position where it left off. If you're fitting a distributed transaction library on top of a normal database, what prevents the DB from rolling back on a crash? – Gili Oct 6, 2008 at 2:47

10 Gili: Because it promised that it would be able to, in the prepare phase of 2PC. – Thomas Aug 26, 2009 at 19:36

4 @Gili The crashed DB doesn't have to replay the transaction. The way traditional DB transactions work is they are actively making those changes in the transaction log, which only occurs if it is able to obtain the necessary row/table/page locks. Essentially after the first phase, the transaction has already modified the DB, but other queries cannot see those changes until they are marked as committed. All that is left to do is mark them as committed. – AaronLS Sep 26, 2014 at 14:55

35

Summarizing everyone's answers:

1. One cannot use normal databases with distributed transactions. The database must explicitly support a transaction coordinator.
2. The nodes are not instructed to roll back because some of the nodes have already committed. What happens is that when the crashed node comes back, the transaction coordinator tells it to finish the commit.

Share Improve this answer

Follow

edited Jun 7, 2019 at 11:46

Oliv

10.8k ● 3 ● 58 ● 78

answered Feb 13, 2009 at 3:49

Gili

89.7k ● 104 ● 410 ● 720

4 What happens if the coordinator crashes while the node is down, and the node wakes up while the coordinator is still down? What does the node do then? – Aviad P. Dec 29, 2009 at 10:50

3 I suggest opening a separate question for this issue. I don't know the answer myself but I'm guessing that the node waits for the coordinator to come back before accepting client connections. – Gili Dec 29, 2009 at 15:37

2 Without the coordinator, everything grinds to a halt of course. – Jaco Van Niekerk Jul 9, 2013 at 10:03

26

No. Point 4 is incorrect. Each node records in stable storage that it was able to commit or rollback the transaction, so that it will be able to do as commanded even across crashes. When the crashed node comes back up, it must realize that it has a transaction in pre-commit state, reinstate any relevant locks or other controls, and then attempt to contact the coordinator site to collect the status of the transaction.

The problems only occur if the crashed node never comes back up (then everything else thinks the transaction was OK, or will be when the crashed node comes back).

Share Improve this answer

Follow

answered Oct 5, 2008 at 12:19

Jonathan Leffler

752k ● 145 ● 946 ● 1.3k

Doesn't this assume that the database is aware of distributed transactions? I thought you are supposed to be able to fit distributed transactions on top of normal databases that know nothing of it... – Gili Oct 6, 2008 at 2:43

7 No; the DBMS have to be aware of their responsibilities in the 2PC protocol, and the key responsibility is to retain a transaction in a Go/No-Go state until the coordinator gives the command. You lose (autonomy) at that point. Informix implements heuristic rollback to deal with vanished systems. – Jonathan Leffler Oct 11, 2008 at 13:56

20

Two phase commit isn't foolproof and is just designed to work in the 99% of the time cases.

"The protocol assumes that there is stable storage at each node with a write-ahead log, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other."

http://en.wikipedia.org/wiki/Two-phase_commit_protocol

Share Improve this answer

Follow

answered Oct 5, 2008 at 12:22

reno

229 ● 1 ● 5

Even if the assumptions you mentioned are met I don't see how the node is supposed to know to replay the transaction (instead of rolling it back) when it comes back from a crash. As far as I understand it, the database isn't aware of the distributed transaction; something on top of it does. – Gili Oct 6, 2008 at 2:45

@Gili: the database has to know about the distributed transaction to support 2PC correctly. If you build something on top, then you don't have a reliable 2PC implementation. – Jonathan Leffler Oct 28, 2008 at 18:50

2 This is not correctly said. 2PC is 100% correct under certain assumptions. For this question, the participant must durably store data from pre-commit and must be able to finish the commit after a restart, if coordinator instructs it to. The coordinator must also durably store the final decision to rollback or commit and retry it, should it fail. Also, all participants must not crash forever, they must eventually restart and have the durably stored data. If these are met,

2PC is 100% correct. If not, it can be incorrect. – Oliv Jun 7, 2019 at 12:09

8

There are many ways to attack the problems with two-phase commit. Almost all of them wind up as some variant of the Paxos three-phase commit algorithm. Mike Burrows, who designed the Chubby lock service at Google which is based on Paxos, said that there are two types of distributed commit algorithms - "Paxos, and incorrect ones" - in a lecture I saw.

One thing the crashed node could do, when it reawakes, is say "I never heard about this transaction, should it have been committed?" to the coordinator, which will tell it what the vote was.

Bear in mind that this is an example of a more general problem: the crashed node could miss many transactions before it recovers. Therefore it's terribly important that upon recovery it should talk either to the coordinator or another replica before making itself available. If the node itself can't tell whether or not it has crashed, then things get more involved but still tractable.

If you use a quorum system for database reads, the inconsistency will be masked (and made known to the database itself).

Share Improve this answer

Follow

answered Oct 5, 2008 at 12:43

HenryR

8,509 ● 7 ● 36 ● 39

1 How does the node replay the transaction when it reawakens? As far as I understand it, normal databases *always* roll back on a crash and they're not aware of distributed transactions (you fit some library on top of them). As such, I don't see how you can make this work. – Gili Oct 6, 2008 at 2:49