Failures in the Two-Phase Commit Protocol

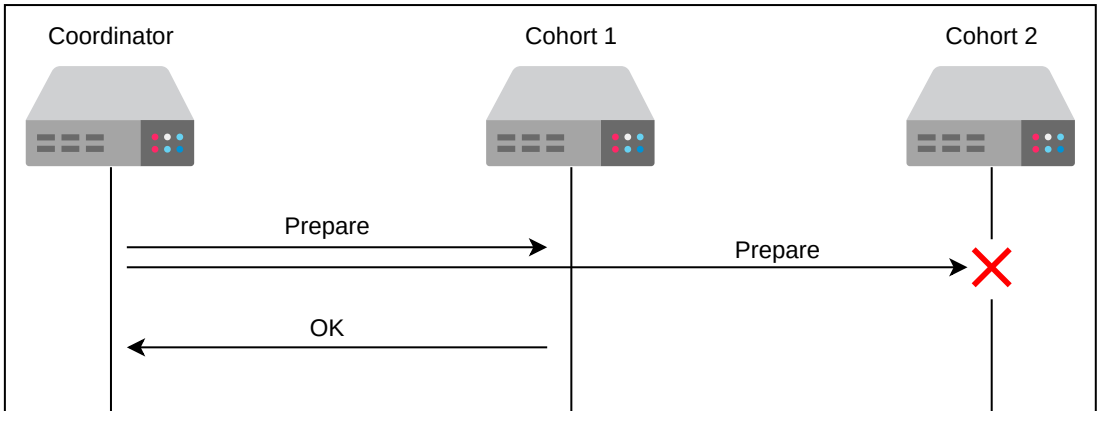
Learn how 2PC behaves under node or network faults.

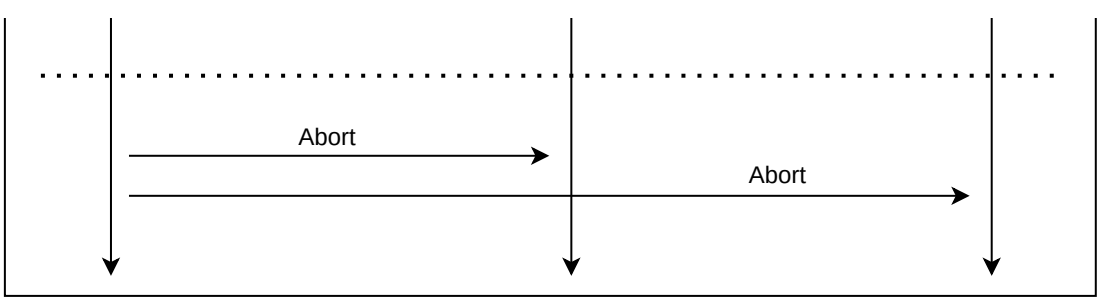
We'll cover the following

In a distributed system, nodes (coordinators or cohorts) can fail anywhere midway through the 2PC transaction. The network can delay or lose messages, or the network can partition the participants. In this lesson, we will learn how 2PC behaves when different faults occur.

Cohort failure

Let's analyze a few failure scenarios. For instance, in the illustration below, if one of the cohorts fails in the prepare phase, the coordinator cannot commit because it needs affirmative votes from all cohorts. So, the coordinator will terminate the transaction if any cohort is unavailable (does not respond before timeout). This requirement adversely affects the system's availability, as the failure of a single node can prevent transactions from taking place.





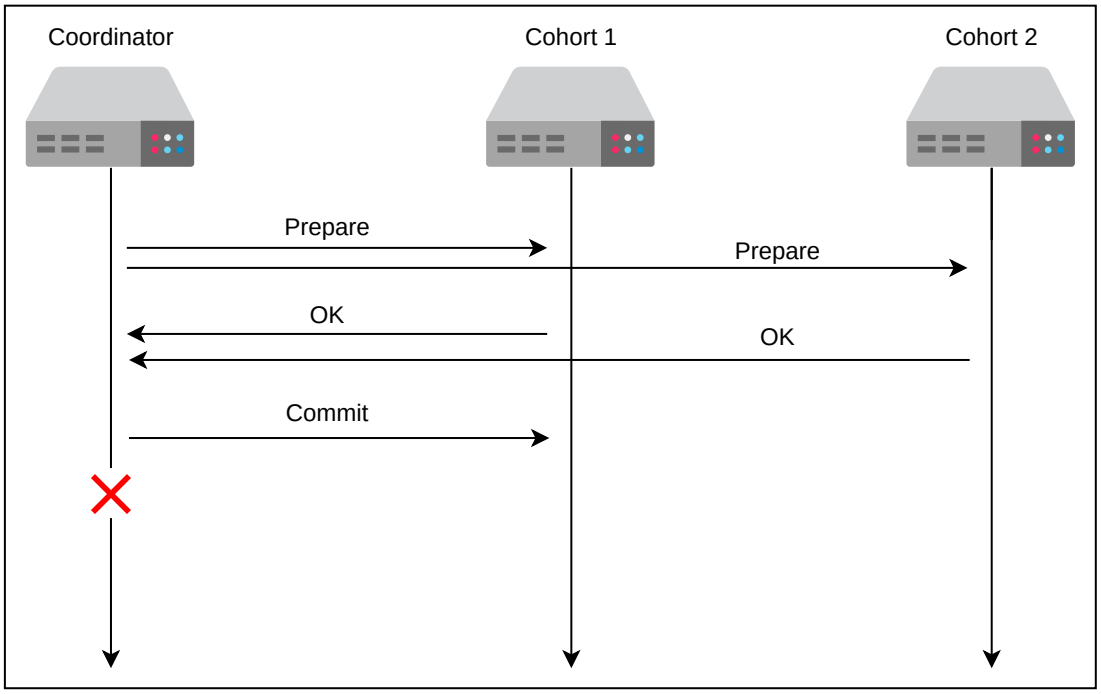
Cohort failure

Coordinator failure

2PC ensures that all transaction cohorts agree on committing or aborting the transaction. However, if the coordinator crashes before sending the commit request (which is after completing the prepare phase but before starting the commit phase, cohorts are left uncertain.

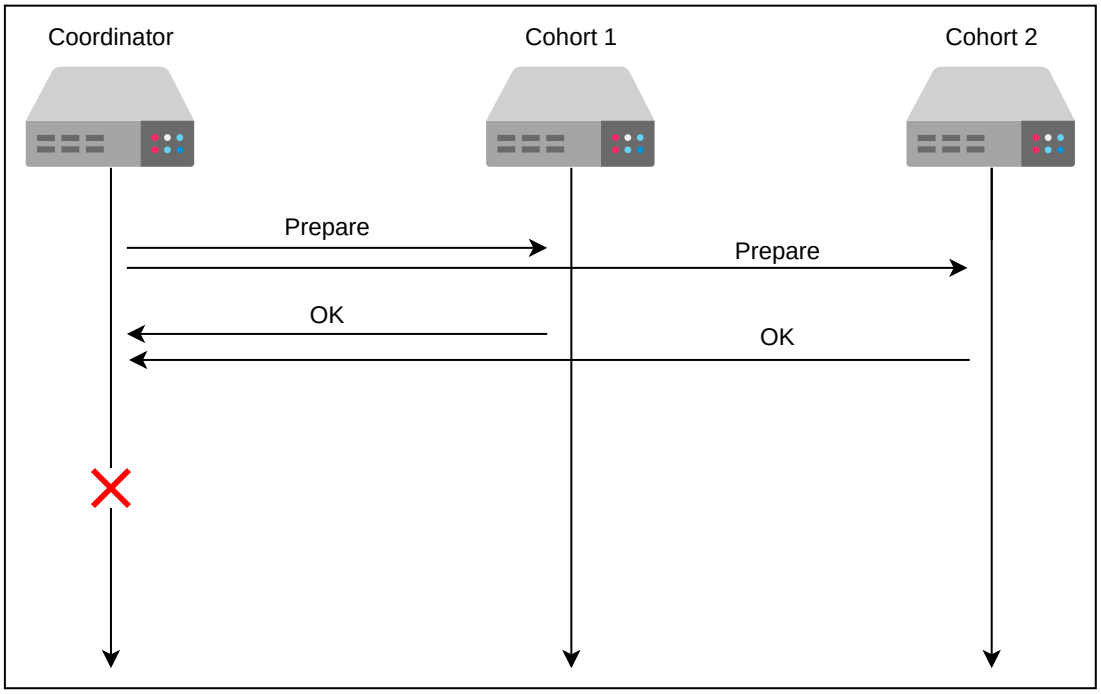
In the event of a coordinator failure before sending prepare requests, cohorts can safely abort the transaction. However, if a cohort has already received a prepare request and voted affirmatively, it must wait for the coordinator's decision on whether to commit or abort the transaction. In case of a coordinator crash, the cohort is left in doubt and cannot unilaterally abort or commit.

The scenario is depicted in the figure below. In this instance, the coordinator is determined to commit, and Cohort 1 gets the commit request. Regrettably, Cohort 2 does not receive the commit request due to the coordinator's crash and cannot determine whether to proceed with the commit or abort. Simply waiting for a timeout is not helpful since a unilateral abort by Cohort 2 would lead to inconsistency with Cohort 1's successful commit. Conversely, committing without coordination is unreliable because other participants may have already aborted the transaction.



Coordinator failure after sending the decision to one Cohort

The above problem can still be managed if Cohort 2 contacts Cohort 1's transaction logs (this is not part of the protocol but can be done with optimization in the system). But there is another problem that cannot be solved with the above optimization: What if the coordinator failed before broadcasting the decision to any cohort? In such a scenario, the whole cluster will be uncertain.



Coordinator failure before sending the decision to any Cohort

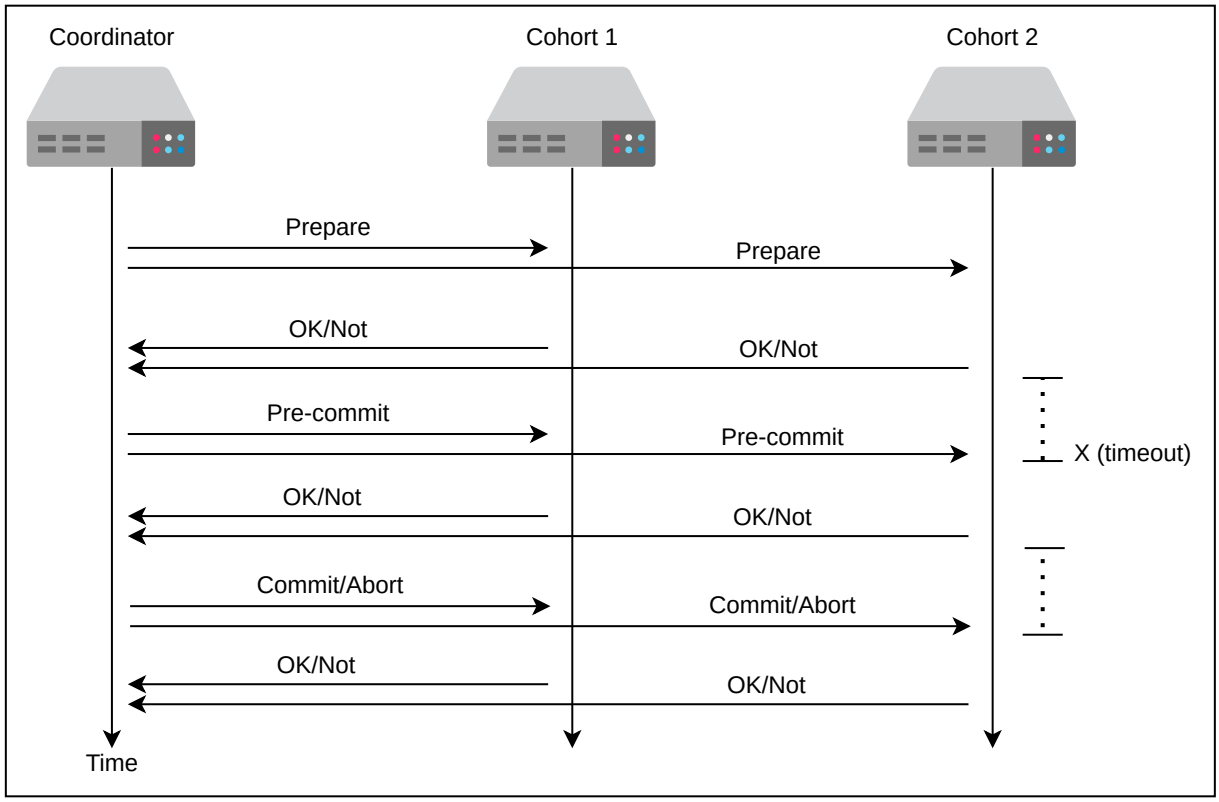
To ensure the consistency of transactions in a distributed system, the coordinator must record its decision to commit or abort in a transaction log on disk before sending requests to participants. After recovering from a failure, the coordinator reads the transaction log to determine the status of all unresolved transactions. In the event of a permanent coordinator failure, the final decision will not be known to cohorts. As a result, the 2PC algorithm is considered a blocking atomic commitment algorithm. If the coordinator does not recover, human intervention is required to select the replacement coordinator. The replacement coordinator must gather votes for the transaction again and make a final decision. Transactions without a commit record in the log are aborted.

The 2PC protocol is categorized as a blocking atomic commit protocol since it can potentially stall while waiting for the recovery of the coordinator. While it is theoretically possible to create a non-blocking atomic commit protocol that doesn't get stuck if a node fails, making it work practically is challenging.

> **Note:** The 2PC protocol always preserves safety conditions (preserves atomicity) but loses liveness (no progress is made when a coordinator or a cohort fails at a specific point in the lifespan of 2PC) under different kinds of faults.

Three-phase commit (3PC)

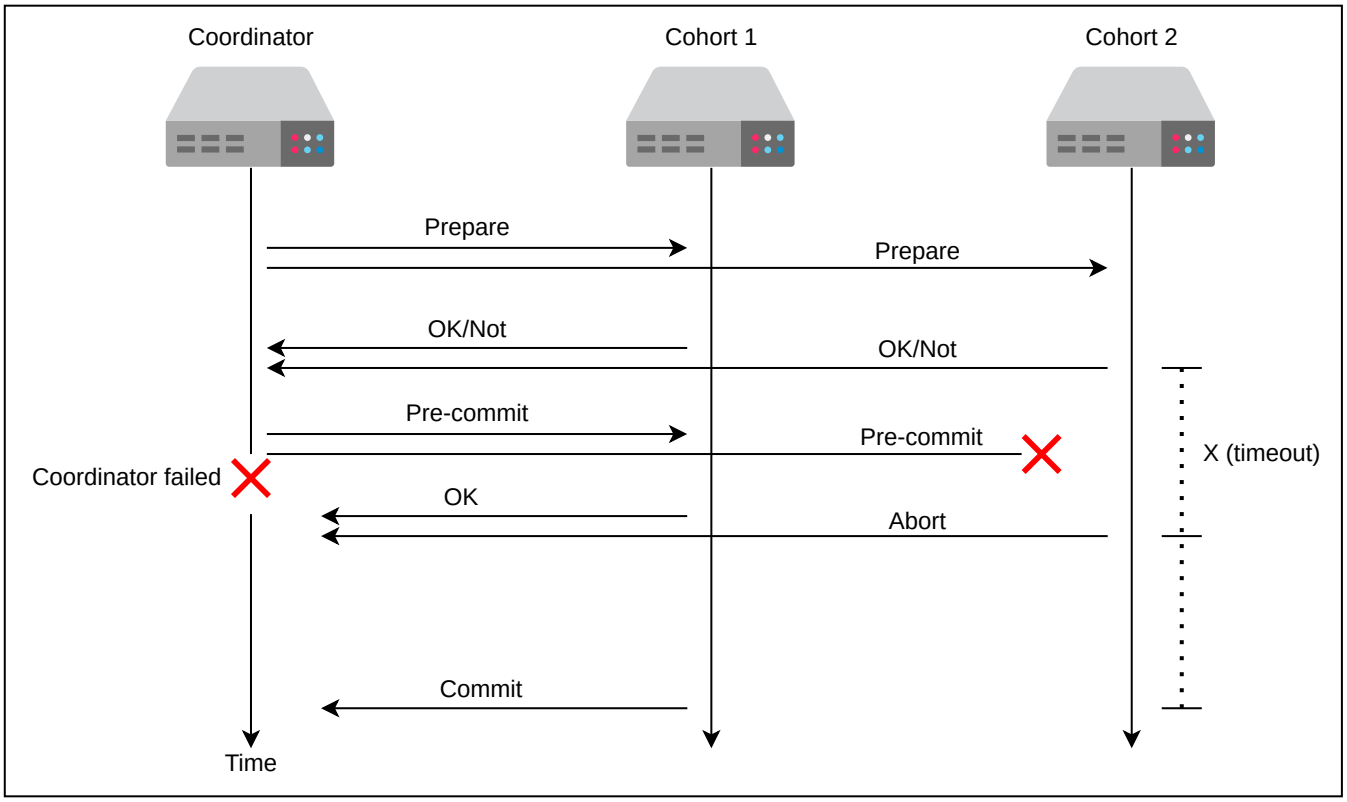
The **three-phase commit (3PC)** protocol was designed to address the limitations of 2PC and make the atomic commitment protocol robust against coordinator failures and undecided states. 3PC assumes a network with bounded delay. In case of coordinator failure, 3PC includes an additional step and timeouts on both sides that can allow cohorts to proceed with either commit or abort, depending on the system state. 3PC is a protocol that builds upon the 2PC protocol. It has an additional phase, called the pre-commit phase. The 3PC protocol consists of the prepare, pre-commit, and commit phases.



Three-phase commit: Cohorts terminated the timer before its expiry as messages from the Coordinator came in before the timeout

In a distributed system, the 3PC protocol coordinates state transitions between the coordinator and cohorts to ensure that all nodes agree on the same decision. Cohorts must wait for the previous phase to be completed by all nodes before moving on to the next phase and can abort the transaction if they do not hear from the coordinator before the timeout. Unlike 2PC, 3PC can recover from coordinator failures and allows cohorts to proceed with a deterministic decision. However, the worst-case scenario for 3PC is a network partition where some nodes successfully pass to the pre-commit state and proceed with the commit. In contrast, others cannot communicate with the coordinator and will abort after the timeout. This results in a split brain, leaving participants in an inconsistent and contradictory state.

> **Note:** In the 3PC protocol, coordinator failures do not block cohorts. If a cohort times out before getting a pre-commit message, a unilateral abort is done. Similarly, if a cohort does not get a commit message, a unilateral commit is done.



Coordinator failure in the pre-commit phase. Cohort 1 unilaterally committed while Cohort 2 unilaterally aborted.

3PC involves a higher message overhead, may introduce potential inconsistencies, and may not perform well in the presence of network partitions. These limitations could be the primary reason why 3PC is not commonly used in practice.

> **Note:** The 3PC assumes a network with bounded delay for timeouts to work correctly. Additionally, 3PC assumes that nodes take bounded time to respond to the messages. On an asynchronous network, messages can take arbitrarily long, and the global internet can act like an asynchronous network. Using 3PC for such scenarios can cause safety hazards where some cohorts have committed while others haven't. Most practical systems use 2PC instead of 3PC because 2PC is always safe, while 3PC is not always safe (it can violate atomicity).

Conclusion

In conclusion, the 2PC protocol is a widely used approach for coordinating distributed transactions and ensuring atomicity. It provides a way for a coordinator to ensure that all cohorts agree on committing or aborting a transaction. Despite some limitations and drawbacks, it remains a popular and effective solution in many distributed systems.

System design wisdom in 2PC design

- When deciding to pick algorithms based on safety and liveness conditions, often it is a better bet to side with safety and let go of liveness under specific fault conditions. It is a tradeoff between strong consistency and availability, a recurring theme in distributed systems. Strongly consistent systems are easier to program and reason about correctness for the end users.
- Google's Spanner uses 2PC over Paxos to reduce 2PC's liveness issues. Additionally, utilizing Google's well-managed network allows 2PC to remain stable. Google Spanner reports that their 2PC takes 150 ms on average and 320 ms at p99 with 200 cohorts. It shows the promising feasibility of 2PC in large-scale systems.
- Correctly understanding the assumptions of an algorithm is necessary. For example, using 3PC in an asynchronous environment will break 3PCs bounded delay assumption, causing it to violate a safety condition (atomicity).
- Consensus algorithms are amongst the trickiest algorithms, with many subtle edge cases. For engineers, it is highly advised to use well-understood algorithms instead of trying to invent new ones. Additionally, even small algorithm tweaks in standard algorithms should be done with abundant caution.