Mina Ayoub

Mar 28, 2017 · 5 min read ·

Listen

Understanding of consistency in distributed systems

First of all, what is consistency?

Consistency is the agreement between multiple nodes in a distributed system to achieve a certain value.

Specifically, it can be divided into strong consistency and weak

consistency.

Strong consistency: The data in all nodes is the same at any time. At the same time, you should get the value of key1 in node A and the value of key1 in node B.

Weak consistency: There is no guarantee that all nodes have the same data at any time, and there are many different implementations. The most widely achieved is the ultimate consistency. The so-called final consistency means that the same data on any node is the same at any time, but as time passes, the same data on different nodes always changes in the direction of convergence. It can also be simply understood that after a period of time, the data between nodes will eventually reach a consistent state.

Distributed and consistent application scenarios:

Multi-node provides read and write services to ensure high availability

and scalability (ZooKeeper, DNS, redis cluster)

Problems faced by distributed systems:

- Message asynchronous asynchronous (asynchronous): The real network is not a reliable channel, there are message delays, loss, and inter-node messaging can not be synchronized (synchronous)
- node-fail-stop: The node continues to crash and will not recover
- node down recovery (fail-recover): Node recovery after a period of time, the most common in distributed systems
- network partition: There is a problem with the network link, separating N nodes into multiple parts
- Byzantine failure (byzantine failure) [2]: Node or downtime or logic failure, even without the card to throw out the interference resolution information

The general premise of distributed system design that needs to meet the consistency is that there is no Byzantine general problem (intranet trusted)

The basic theory of distributed systems, the FLP theorem, is mentioned here, that is, when only the node is down, the availability and strong consistency cannot be satisfied at the same time. Another point of view is CAP theory, namely strong consistency, availability and partition fault tolerance, only two of which can be guaranteed.

There are many agreements to ensure consistency, including 2PC, 3PC, Paxos, raft and PacificA.

2PC:

2-stage lock commit protocol to guarantee the atomicity of operations on multiple data slices. (distributed transaction)

The nodes are divided into

coordinators and participants (participat), and the execution is divided into two phases.

- Phase 1: The coordinator initiates a proposal to ask whether each participant is accepted. Participate performs transaction operations, writes undo and redo information to the transaction log, and replies yes or no to the coordinator.
- Phase 2: The coordinator submits or aborts the transaction according to the feedback of the participant. If the participant is all yes, it is submitted, as long as there is a participant reply no. Participate formally commits or terminates the transaction according to the commit/Rollback information of the coordinator, and releases the occupied resources and returns ack.

Advantages: simple principle and easy implementation

Disadvantages: synchronous blocking, single point problem, data inconsistency (coordinator crashes

before sending commit request or network causes part of the consensus does not receive commit, then part of the participant cannot commit transaction), too conservative (if participant is in coordination) If there is a failure during communication, the coordinator can only rely on the timeout mechanism to determine whether the transaction needs to be interrupted.

3PC:

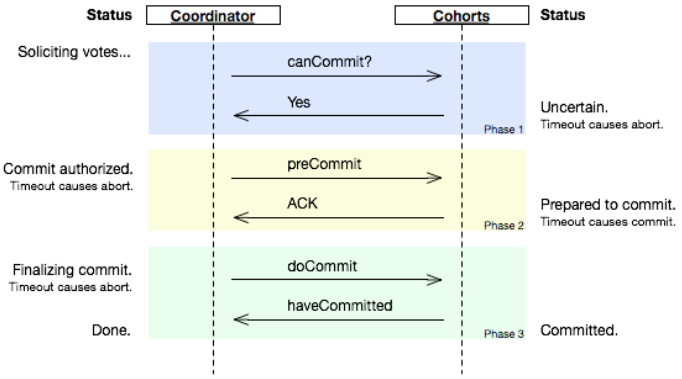
3-stage lock submission protocol to guarantee the atomicity of operations on multiple data slices. (distributed transaction)

Relative to 2PC, divided into inquiry, pre-submission, and submission of 3 stages (resolving blocking, but it is still possible that data is inconsistent)

Process: After receiving the participant's feedback (vote), the coordinator enters phase 2 and sends a prepare to commit command to each participant. The participant can

Mina Ayoub
824 Followers
I'm enthusiastic about being part of something greater than myself and learning from more experienced people every time I meet them.

lock the resource after receiving the instruction to submit, but requires the relevant operation to be rolled back. After receiving the acknowledgment (ACK), the coordinator enters phase 3 and commit/abort. Phase 3 of 3PC is no different from phase 2 of 2PC. Coordinator watchdog and status logging are also applied to 3PC.



Paxos algorithm (solving single point problems)

The Paxos algorithm is currently the most important consistency algorithm, and all consistency algorithms are a simplified version of paxos or paxos.

The Paxos algorithm resolves multiple values of the same data to agree on a value. The theoretical basis for proof

of correctness: the intersection of any two legal sets (sets consisting of more than half of the nodes) is not empty.

Character:

There are three roles involved in the proposal to the voting process:

- **Proposer:** There may be more than one proponent, and it is responsible for proposing the proposal.
- **Acceptor:** There must be more than one recipient. They vote on the specified proposal, agree to accept the proposal, and disagree.
- **Learner:** Learners, collect proposals accepted by each Acceptor, and form a final proposal based on the principle of minority majority.

In fact, a component in a distributed system can correspond to one or more roles.

Algorithm Description:

*** The first stage (Prepare stage)**

Proposer:

- Select proposal number n and send a prepare request with number n to most Acceptors.

Acceptor:

- If the proposal number n received is larger than the number already received, Proposer promises not to accept proposals with a number less than n. If the proposal has been accepted before, the proposal with the highest number among the accepted proposals will be The number is sent to Proposer.
- If the proposal number n received is less than the maximum number of proposal numbers it has received.

*** The second stage (Accept stage)**

Proposer:

- First, the response is received, one by one:
- If a rejection is received, it will not

be processed.

- If you receive the consent and also receive a proposal that Acceptor has accepted, make a note of the proposal and the number.
- After processing the response, count the number of rejections and consents:
- If most reject, prepare for the next proposal.
- If most agree, select the proposal with the largest proposal number as the proposal from the proposals accepted by these Acceptors, and use the proposal without the own, and send the Accept message to the Acceptor one by one.

Acceptor:

- If the proposal number n received is less than the maximum proposal number it has received.
- If the proposal number n received is equal to the maximum proposal number it has received, the proposal will be accepted.
- If the proposal number n received is greater than the maximum

proposal number it has received.

- Form a consensus (parallel to the Prepare&Accept phase)

Acceptor:

- Whenever a proposal is accepted, the proposal and number are sent to Learner.

Learner:

- Record the proposal currently accepted by each Acceptor. If an Acceptor sends multiple proposals in succession, the proposal with the highest number is retained.
- Count the number of Acceptors accepted for each proposal. If more than half of them are accepted, a consensus will be formed.