12. Split Brain

Let's learn about split-brain and how to handle it.

We'll cover the following

Background#

In a distributed environment with a central (or leader) server, if the central server dies, the system must quickly find a substitute, otherwise, the system can quickly deteriorate.

One of the problems is that we cannot truly know if the leader has stopped for good or has experienced an intermittent failure like a stop-the-world GC pause or a temporary network disruption. Nevertheless, the cluster has to move on and pick a new leader. If the original leader had an intermittent failure, we now find ourselves with a so-called **zombie leader**. A zombie leader can be defined as a leader node that had been deemed dead by the system and has since come back online. Another node has taken its place, but the zombie leader might not know that yet. The system now has two active leaders that could be issuing conflicting commands. How can a system detect such a scenario, so that all nodes in the system can ignore requests from the old leader and the old leader itself can detect that it is no longer the leader?

Definition#

The common scenario in which a distributed system has two or more active leaders is called split-brain.

Split-brain is solved through the use of **Generation Clock**, which is simply a monotonically increasing number to indicate a server's generation.

Solution#

Every time a new leader is elected, the generation number gets incremented. This means if the old leader had a generation number of '1', the new one will have '2'. This generation number is included in every request that is sent from the leader to other nodes. This way, nodes can now easily differentiate the real leader by simply trusting the leader with the highest number. The generation number should be persisted on disk, so that it remains available after a server reboot. One way is to store it with every entry in the Write-ahead Log.

Examples#

Kafka: To handle Split-brain (where we could have multiple active controller brokers), Kafka uses '**Epoch number**,' which is simply a monotonically increasing number to indicate a server's generation.

HDFS: ZooKeeper is used to ensure that only one NameNode is active at any time. An epoch number is maintained as part of every transaction ID to reflect the NameNode generation.

Cassandra uses generation number to distinguish a node's state before and after a restart. Each node stores a generation number which is incremented every time a node restarts. This generation number is

included in gossip messages exchanged between nodes and is used to distinguish the current state of a node from the state before a restart. The generation number remains the same while the node is alive and is incremented each time the node restarts. The node receiving the gossip message can compare the generation number it knows and the generation number in the gossip message. If the generation number in the gossip message is higher, it knows that the node was restarted.

← Back

Next →

11. Phi Accrual Failure Detection

13. Fencing

Mark as Completed

Report an Issue