03/20 : System Patter - Quorum - Background

1

Background

In Distributed Systems, data is replicated across **multiple servers** for **fault tolerance** & **high availability**. Unfortunately, it also introducing another challenge here, how to make sure that **all replicas are consistent and updated**.

Meaning, does they all have the latest copy of the data and that all clients see the same view of the data?

Now, let's understand the **problem statement.**

In distributed system if we wait for all the replica's servers updated, and than mark operation successful, it might not help in achieving especially low latency operation or High availability of the system.

So how to solve or negotiate for consistency and availability somewhat at middle ground level.

one point I would like to mention here that as per the CAP Theorem **which is also considered as a System Pattern**, C is for Consistency, A is for Availability, P is for Partition Tolerance, In distributed system, Partition tolerance is unavoidable, so it must be present.

Now, out of Consistency & Availability you will have to choose one over other as per your system requirement. It don't mean that you should not having that feature in your system, like Eventual consistent system must have high availability and fully partition tolerant.

2

Ok, let's comeback to our topic rightnow because anyway I will explain in more detail about pattern, while sharing my view on CAP.

So, **In distributed environment, a quorum is the minimum number of servers or nodes on which a distributed operation needs to be performed successfully before declaring the operation's overall successful.**

Meaning, you are creating some quorum for balancing the strike between consistency and availability.

Because sometime your system might don't want to leave consistency and Availability or performance/latency etc.

So, striking the balance between these core features and producing the acceptable features for your system is primarily goal of Quorum pattern.

And this is the **architectural decision you are taking as a system architect**.

Another great use case of this pattern is inside consensus algorithm implementation.

If you remember **Chubby**(Distributed Locking Service, from **Google**) who uses **Paxos algorithm for Leader election**, and **Paxos using Quorum inside**.

Similarly, many **distributed system uses Raft algorithm** for consensus and internally this algorithm, also usage Quorum for leader election. As an example of Raft Algorithm, Apache Zookeeper, **YugabyteDB** etc.

Quorum – How?

3

How? Solution-

Suppose a database is replicated on five machines.
In that case, quorum refers to the minimum number of machines that perform the same action (commit or abort) for a given transaction in order to decide the final operation for that transaction.

So, in general Quorum work on Majority. And majority means here more than half,
So, if you have replication factor N(05) than quorum will work optimally for 03 machines Quorum. Meaning if our 03 machine have updated data than as per quorum definition, operation could be considered successful.

What value should we choose for a quorum?
So, the Formula is **(N/2+1)** and
If you talk about both read and write than it should be - **R+W > N**

N = nodes in the quorum group
W = minimum write nodes
R = minimum read nodes

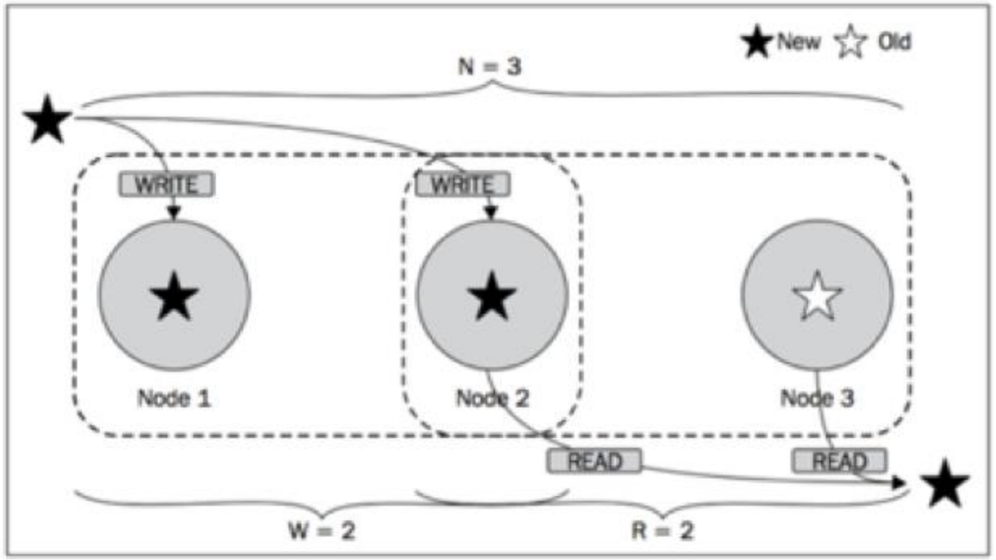
If a distributed system follows R + W > N rule, then every read will see at least one copy of the latest value written.

For example, a common configuration could be (N=3, W=2, R=2) to ensure strong consistency. Here are a couple of other examples:
- ✓ (N=3, W=1, R=3): fast write, slow read, not very durable
- ✓ (N=3, W=3, R=1): slow write, fast read, durable

So, one best strategy for us as a thumb rule for Best performance (throughput/availability) when 1<*r*<*w*<*n*, because reads are more frequent than writes in most applications

4

Consistent reads after write success



Apache Cassandra Quorum Writes – Success Case