Highly Available Databases

Features:

Amount of data we need to store: 100 TB
Support for updates: Yes
Can the size of the value of the key increase with updates ? Yes, at some point, the whole data won't fit in a single machine. Also, let's assume the upper cap for a single value will be 1 GB (To fit in a single machine)
QPS For this DB: 100 K

Estimations:

Total size = 100 TB
Estimated QPS: 100 K
Total Number of Machines to store the Data: 10 (each with 10 TB Hard disk space), Number will be more if we consider replication.

Design Goals:

Is Latency a very important metric ? Since we want to be available all the time, we should try to have lower latency.

Consistency vs Availability: As the question states, we need good availability and partition tolerance. Going by CAP theorem, we need to compromise on consistency if we go with availability and partition tolerance. We can however aim for eventual consistency, as is the case with storage systems, data loss is unacceptable.

Deep Dive:

Is sharding required? Yes, 100 TB can't fit into a single machine, even if we find a machine that can store this data, that machine alone has to handle all queries - so, there will be a significant hit on performance, at the same time, scalability is difficult. So we need sharding here, distribute data and load into multiple machines.

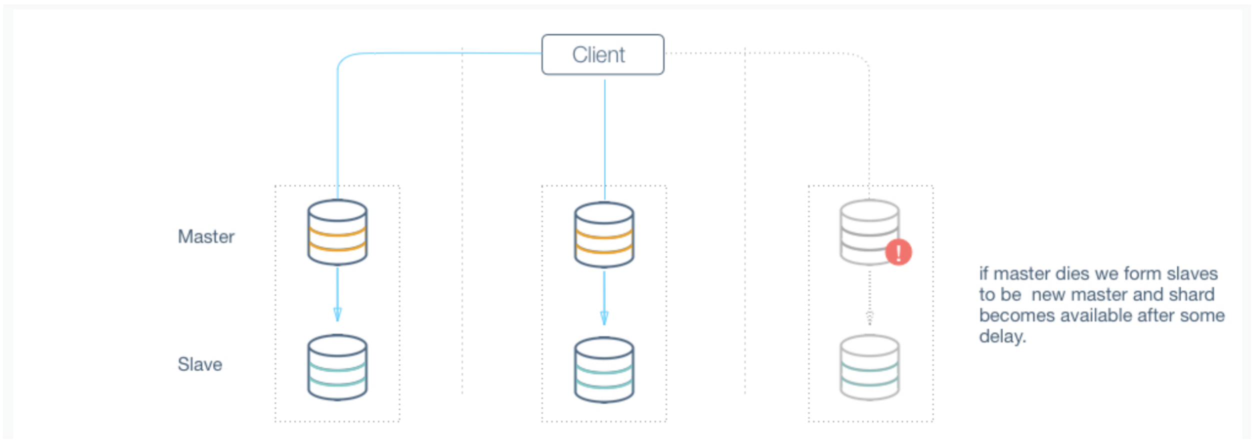
Should data stored be normalized ?

If the data is normalized, then we need to join across tables and across rows to fetch data. If the data is already sharded across machines, any join across machines is highly undesirable (High latency and less indexing support)
With storing denormalizing info, we might store redundant data but the data related to same row will be in a single machine - this would lead to lower latency. However, if sharding criteria is not chosen properly, it could lead to consistency concerns . For this case, since we are considering

availability and it’s okay to have eventual consistency. Hence, having denormalized rows makes sharding easier and suits our use case better.

How many machines per shard ? How does read/write look in every shard ?
Let’s assume we somehow sharded the rows into shards.
Possible Architectural designs within the shard:

Master Slave:


Client
Master
Slave
if master dies we form slaves to be new master and shard becomes available after some delay.

Slave reads all new updates from master and tries to get eventually consistent.
Client reads either from master or slave depends on which responds quicker (slightly better approach would be to give first preference to master because slave might not be in full sync yet - could lead to inconsistent views on newer entries but this ensures high read availability)
Writes will go to master, but in case of a failure - the slave becomes the new master (however , there is a chance of loss of data for some period until the slave is fully up) - we definitelt need more than machine that should take the writes.

Multi Master:

Both machines accept read and write traffic. If M1 accepts a write without depending on M2, this could lead to consistency issues. DBs also get operations out of order which could cause eventual inconsistency. This also will not solve our problems.

Peer to Peer Systems:

Can peer to peer systems be highly available in case of a DB machine dying ?

Yes, we define a peer to peer system where every node is equally privileged and any two nodes can communicate. So the system is theoretically available even if a DB machine crashes. Cassandra and Dynamodb are examples of such systems, both of them lack master nodes and hence no single point of failure.

How will we exactly shard data for peer to peer system ?

https://www.interviewbit.com/problems/sharding-a-database/