Phu Lam
Homework 1
CSC 369

By default, Redis uses asynchronous replication with a single leader, which is low latency and high speed. Replicas asynchronously acknowledge writing periodically with the leader. So the main leader doesn't wait every time for a write request. However, it knows what replica did process which command. This provides optional synchronous replication.

Redis does not support multi-leader and Leaderless replication. But, synchronous replication can be requested by clients. But, it does not become a fully consistent partition database. Users are only able to configure there are specified numbers of acknowledged write to other replicas. Therefore, acknowledged write requests can still be lost during the leader failure. However, the chance of losing write after a failover event is greatly reduced.

When the connection link between the leader and replica is lost, it will attempt to do a partial resynchronization, meaning replicas will try to obtain the part of the stream of data it missed during the disconnection. Furthermore, if partial resynchronization fails, it will involve a process similar to adding a new follower in which it creates a snapshot of all its data, then sends it to the replica and continues sending a stream of data as the dataset changes.

There are two types of partitioning in Redis: Range partitioning and hash partitioning. In range partitioning, the strategy is accomplished by mapping ranges of objects into specific instances. For instance, we specify that events from ID 0 to 1000 will be placed into partition P1 and events from ID 1001 to 2000 will be placed into partition P2, and so forth. In harsh partitioning, the strategy is accomplished by a hash function (eq. Modules function) which will hash the key into a number then data is stored in a different-different partition.

Redis uses a partition rebalancing strategy which is similar to the method we went over in class. For example, if we have three new nodes A, B, and C and we want to add a new node D, we need to move some hash slots from nodes A, B, and C to D. Similarly, if we want to delete node B, we can just move the hash slots in B to A and C. We can remove B from the cluster when B is empty. When moving hash slots from one node to another does not need to stop any operations; Therefore, there is no downtime.

Works cited

"The Redis Manual." *Redis*, https://redis.io/docs/manual/.