NoSQL Databases

Principles

Distribution Models

- Generic techniques of data distribution
    - sharding
        - different data chunks is put on different nodes
            - data partitioning
        - motivation: increases performance
    - replication
        - same data is copied on multiple nodes
        - motivation: increases fault-tolerance
- We can use either of them or combine them
- Distribution model = is a specific way to do sharding, or replication or combination of both
- NoSQL databases often offer automatic sharding & replication

Distribution Model: Single Server

- We call this setup standalone
- Running a database on a single machine spares a lot of problems
- Running NoSQL on single server still can make sense
  - if cluster is needed; we can scale when needed
  - even one node, we can get other benefits; flexibility of schema

Sharding (Data Partitioning)

- Placing different data on different nodes
- different data meaning depends on the underlying database type
  - for example key-value database
    - different data means different key-value pairs
  - in document database
    - different data means different documents
- Related pieces of data that are related to each other should be stored physically together

Sharding (Data Partitioning)

- Try to ensure that:

  1. data accessed together should be kept together

     - so the user gets all related data from single node; instead of collecting from several nodes on the cluster

     - operation involving data on multiple shards should be avoided

     - this is achieved with data aggregates

  2. data arrangement on nodes

     - try to keep the load balanced

  - Many noSQL databases offer auto-sharding

  - A node failure means that shards on that node becomes unavailable

    - therefore, sharding is often combined with replication

Replication

- Replication
  - placing multiple copies (replicas) of the same data on different nodes
  - replication factor = number of replicas
  - two approaches
    - master-slave architecture
    - peer-to-peer architecture

Master-slave replication

- Architecture
    - one node is the primary node (master)
        - is responsible of data management
        - process all data updates
        - reads from any node
    - all other nodes are secondary (slaves)
        - keeps the data

Master-slave replication

- Suitable for read-intensive applications
- To scale
  - more reads requests —> add more nodes
- Limited by ability of the master to handle update operations
- In case the master fails, a new master will be appointed
  - manually (user-defined) or automatically (cluster-elected)
- consistency
  - enough, at most one write request is handled at a time
  - master propagate updates to replicas on slave nodes
  - no read happens until they finish (synchronization)

Peer-to-Peer Replication (Architecture)

- No master
  - all nodes are equal, have equal roles & responsibilities
- both read & write can be handled by any node
- so no single node of failure or bottleneck
- both read and write operations scale
  - more request —> deploy more nodes
- consistency
  - multiple write requests can be handled at a time
  - so to avoid conflict, synchronization is required

Sharding & Replication (1)

- Number of replicas
  - replication factor = number of replicas
  - replication factor does not have to be equal to the number of nodes
  - 3 replicas is a good replication factor

Sharding & Replication (2)

- Sharding & master-slave architecture
  - each data shard is replicated
  - a node can be a master for some data but slave for other

Sharding & Replication (3)

- sharding & peer-to-peer architecture
  - it is common strategy used by column-family databases
  - typical default replication factor is 3
    - each shard is placed on three nodes
  - there is no master replica
  - so we need consistency approach
    - consistency is the lack of contradiction in the DB

Sharding & Replication (4)

- Any distribution model should deal with the following questions
  - can all nodes serve read and write requests?
  - which replica placement strategy is used?
  - level of consistency & availability?

Summary / Data Distribution

- Improving performance
  1. put the relevant data close to each other, and if you have a world wide datacenter, put data close to where it will be accessed
  2. try to keep load balanced among nodes

Summary / Master-slave Replication

- Consistency is not an issue
  - at write, all operation goes via the master
  - at read, the node assigned as a master replica will response, access can be from any replica

Summary / Replication with Peer-to-Peer

- All replicas are equal, no master replica
- All node can support write operation
- Consistency is an issue
  - slow propagation of changes to copies on different nodes
    - inconsistent on read if the same changes is not propagated to all nodes
  - updating different copies of the same data can happen at the same time
    - result in write-write conflict