How Discord Stores Billions of Msgs?

Subscribe YouTube channel for more such content - MsDeep Singh

Database in use for msg storage - Cassandra.

Why choose Cassandra?

① Read/write pattern - 50/50

② Patterns basis type of Discord server - Random Reads

Voice Chat Heavy	Private Text Chat Heavy	Large Public Discord Server
- very few msgs - Approx 1k msgs/year - Returning msgs to user results in many random seeks on disk causing disk cache evictions.	- good No of msgs - 100k to 1M msgs/year - As numbers of members are low, the rate at which data is requested is low — so data will not be present in disk cache	- huge no of msgs as large pool of msgs - Multi million msgs/year - msgs are requested often. so usually present in disk cache.

Keeping Requirements in mind to choose a database

① Linear Scalability – Scalable solution for future traffic and don't require manual re-sharding.

② Automatic Failover – Nodes failing should auto recover.

③ low maintainance – should not require extra setup.

④ Proven to work – Tech should already be in use by other industries.

⑤ Good Performance – Can database work without any caching support on top of it?

⑥ Blob storage is not a solution – requires extra work to deserialize blobs and append msgs to them.

⑦ Open Source – No dependency on third party.

Database meeting the requirements – Cassandra

- Cassandra stores related data contiguously on disk providing minimum seeks and easy distribution around the cluster.

Data Modeling

cassandra is KKV store

- K → primary key
- V → value

K K

- First K → partition key [on which node data will be kept]
- Second K → Clustering key [primary key within partition and helps in data sorting]

```
CREATE TABLE messages(
    channel-id  bigint,
    message-id  bigint,
    author_id   bigint,
    content     text,
    PRIMARY KEY((channel-id, bucket), message-id)
) WITH CLUSTERING ORDER BY (message-id DESC);
```

→ message-id is a snowflake - Twitter library helps in generating roughly sorted 64 bit ids

→ Composition of timestamp, worker number (chosen at startup by zookeeper) and sequence no. (per thread)

→ Cassandra supports partitions till 2GB (you'll see warnings if partition > 100 MB). Large partition should be avoided, why?

- They put lot of GC pressure during compaction, cluster expansion.
- Data in large partition cannot be distributed around the cluster.
- To make sure it is <100MB, data is bucketised
  ↳ it is derived from messag_id.
  ↳ view sample code on shared blog link.

How will you query?

① Generate a bucket from current time to channel-id (channel-id is also a snowflake, also older than first msg)

② Sequentially query msgs for certain threshold of msgs.

Identified issues After launch?

↳ author_id is null.
It should not be null, it is a required field

↳ Eventual consistency is culprit.

① Cassandra is AP database.

Consider a scenario where for msg with message_id "abcdef"
- userA edits it.
- userB deletes it. > at same time.

Cassandra writes are upserts — you'll end up a row with all data missing except primarykey & text.

Two resolutions –

① write whole msg back when editing the msg. – more chances of conflicts for concurrent writes.

② Figure out msg is corrupt & delete it.

↳ Delete the msg if required column is null – author-id

↳ This write operation is slow – Cassandra is eventually consistent and don't delete data immediately. Delete operation also need to be replicated to other nodes as per requirement (even if nodes are temporarily unavailable)

↳ Cassandra achieves this by treating deletes as form of write called "tombstone".

↳ It can skip tombstone on read operation.

↳ Tombstone live for 10 days (can be configured).

In summary, only write if non-null values.

To reduce issues with tombstones – scenario of too many deletes

↳ reduce lifespan of tombstone. For Discord – 2days

↳ Track empty buckets and ignore them.

Read full Blog on Discord Engineering Blog

↳ find link in video description.

Happy Learning