Essay on NoSQL: Twitter's Architecture from a scalability perspective

NoSQL in contrast to SQL, is a non-relational and no-schema database which has the capacity to store large datasets in either key-value pairs or documents. Examples of NoSQL databases are: Hadoop, MongoDB and Apache HBase etc. Likewise, Twitter uses Redis as their backend database. Redis is an open-source key-value in-memory store, which supports a variety of data structures including: lists, sets, stored sets, bitmaps etc. (Redis Labs, 2018).

Here are some of the features Redis provides as a DaaS (Database as a Service):

Redis Enterprise for IoT: Redis makes it easy to collect and process large volumes of data from any device sensors.

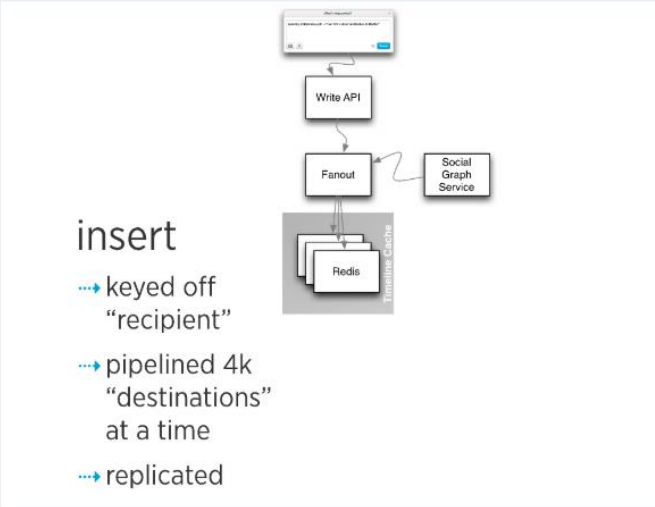
Sharding: Redis allows data to distributed across multiple Redis instances.

Speed: Redis stores the whole dataset in primary memory which makes it fast. It loads up to 110,000 sets/second and 18,000 GETs can be retrieved from an entry level Linux box.

Performance: Changes to data are asynchronously saved on disk using flexible policies based on elapse time/number of update since last saved.

The reason why Redis is used by Twitter is to handle caching user's timelines, tweets and more. Caching is an important aspect of Twitter's as it protects their backing stores from heavy traffic. (Mazdak Hashemi 2017). In addition, for scalability purposes users are only cached if they are active users. Twitter do not waste resources on inactive users so that it can manage resources more efficiently.

Regarding the send tweet mechanism, the tweet comes through the write API that handles writes and triggers the fanout algorithm. The algorithm takes the tweet the user has written, finds the Redis instance that contains the user's tweets and followers and inserts the data.



This is a diagram explains the steps in which a tweet is stored it adopted from (Raffi Kirkorian, 2012).

Regarding partitioning, the fan out takes the social graph service, for e.g. a user's tweets, the user needs to find all the users that are followers of that user, so 20, 000 of them, to be imported into their Redis instances. The followers are pipelined for the process to be done as fast as it can, so approx. 4000 users are done at one time so that way it can be parallelized (Raffi Kirkorian, 2012). One specific fanout instance will take followers 1 - 4000 users and insert them into the timelines whilst simultaneously the fanout instance will take followers 4001 - 8000 and begin inserting them into the timelines as well. This is done to improve the speed of the fanout. It is replicated through the data centers, so whenever an insertion occurs into someone's Redis instance, the insertion occurs in 3 other instances in the data centers. For Cross BC, the entire write path is replicated into secondary and tertiary data centers and all the fan out is replicated over.

Overall, Redis is good choice for Twitter as it provides them with the ability to retrieve data in real-time which provides good user experience.

Bibliography

Raffi Kirkorian (2012), Real-Time Delivery Architecture at Twitter. Available at: https://www.infoq.com/presentations/Real-Time-Delivery-Twitter (Accessed on: 25th Oct 2018)

Redis Labs (2018), Why Redis? Available at: https://redislabs.com/why-redis/ (Accessed on: 11th Nov 2018)

Features of Redis. Available at: https://www.javatpoint.com/features-of-redis (Accessed on: 12th Nov 2018)

Top 5 Redis use cases. Available at: https://www.objectrocket.com/blog/how-to/top-5-redis-use-cases/ (Accessed on: 12th Nov 2018)

Mazdak Hashemi (2017), The Twitter Infrastructure Behind Twitter.https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html (Accessed on: 12th Nov 2018)