Partitioning and Replication in Twitter Architecture

Twitter, which operates on massive data, cannot store its database in a single computer. The best way to handle it efficiently is to shard the whole data and store it across several computers. Sharding strategy involves two main techniques:

1. Partitioning
2. Replication

What exactly does Partitioning mean?

In Partitioning, the data is divided into small chunks and stored across many computers. Each of those chunks is small so that data can be handled by that computer efficiently. Each partition has its own index. There is a broker which forwards the query request to the partition indices and then co-ordinates the partition results accordingly to serve the client.[1] [2]

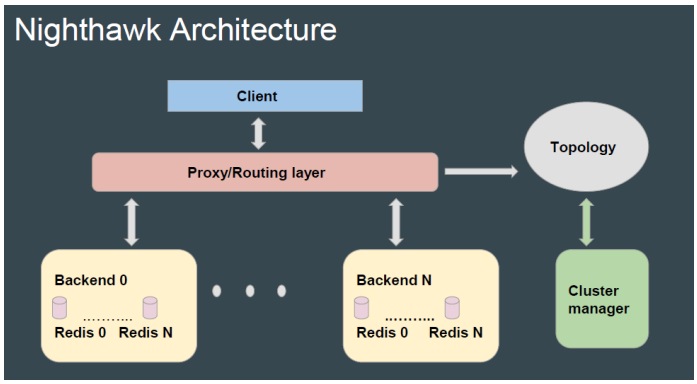
Role of Replication:

Multiple copies of the same data are stored in many systems to avoid single point failure at any case.When there is a data loss in any one of the partition, we can tolerate the loss by providing the data from the other replica we have. At any point of time, Twitter Topology ensures that it has two replicas of the same data maintained in the partitions. [1]

Nighthawk:

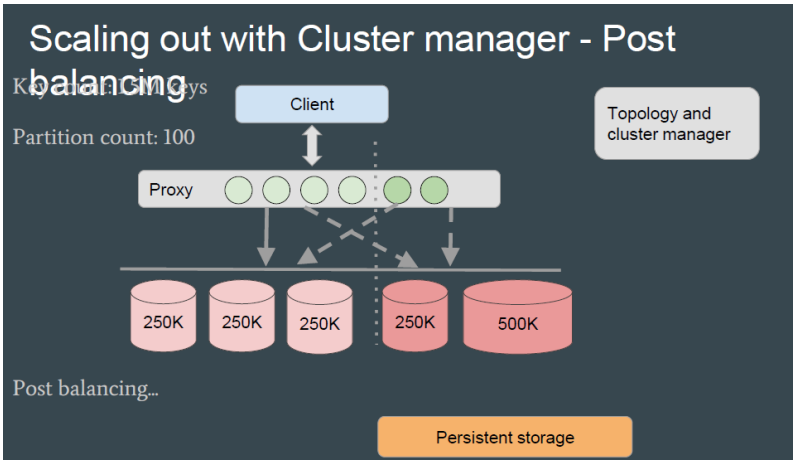
In Twitter, the Sharding mechanism is provided by Nighthawk. Nighthawk is the distributed cache service that runs at Redis.

1. There are many backend database and multiple Redis instances in each to make.
2. Proxy layer routes the request from application level to database.
3. Topology is aware that each partition is associated to which of the Redis node at the backend.
4. Cluster manager manages the partition and replicas in the cluster.



Nightawk Architecture

Instead of 'consistent hashing' and 'homogeneously sized' partitions, Twitter uses '**heterogeneously sized partitions**' that helps to identify the hotspots and reduce the cache misses when we add new partitions in the future. By Hotspotswe refer to the frequently accessed hashtags or keys into one Hotspot partition so that it is easy to fetch them quickly. Here we have 4 partitions of size 250 and one more partition of size 500 in this example. [6]



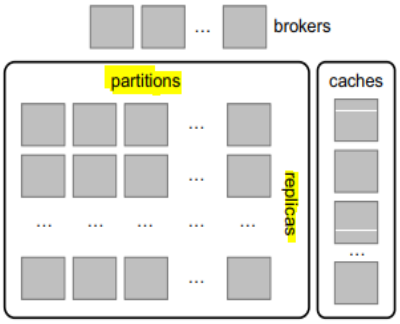
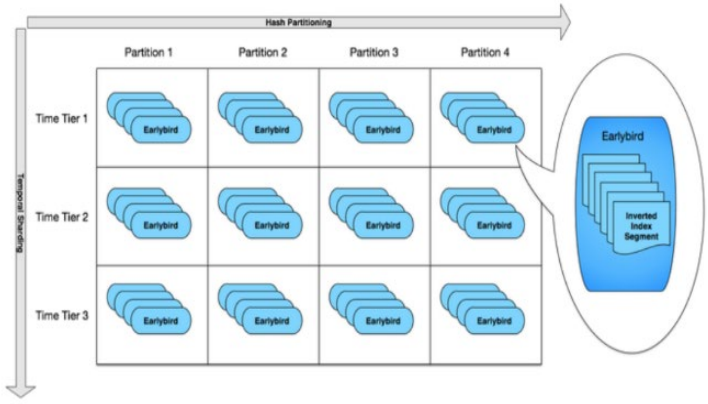


Figure 1: Illustration of a standard distributed search architecture that takes advantage of partitioning, replication, and caching.

Basic Outline of Partition and Replica [4]



Detailed partitioning and Replication [5]

From the basic and detailed Shared architecture in Twitter, we could infer the below analysis [5]:

Sharding Techniques	Description
1. Temporal Sharding	There is partitioning based on time into multiple time-tiers
2. Hash Partitioning	There are several partitions based on hash functions in each time-tier
3. Earlybird	With each hash function partition, data is divided into chunks in earlybird instances.
4. Replicas	Each Earlybird is replicated to maintain the throughput and avoid single point failure.

Earlybird Shards:

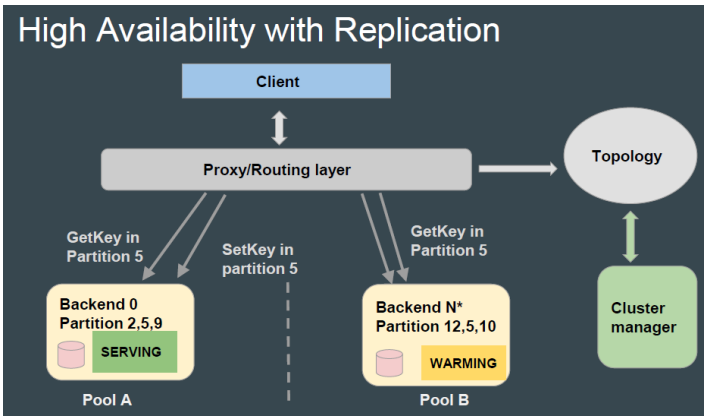
To be more precise, Earlybird shards internally perform '**Inverted Index Full Text operation**' to serve user requests in search operation. When user tweets, the tweet is tokenized, indexed and stored in several replicas of the Earlybird instances in the Redis cluster.

When the user sends the request, query is sent to all the nodes in the cluster and the response is gathered across several nodes having the corresponding search tweet query index. Thus it follows a strategy called '**Scatter and Gather' partitioning approach** [3].

Replicas:

Gizzard in Twitter supports '**Synchronous Best-Effort Replication'.** By synchronous, it means that it maintains two replicas of data at anytime. Twitter maintains the Replication factor as '2'. And by 'Best Effort', it means they support '**Idempotent and Cumulative**' operations to ensure consistency of Replication. [6]

Serving and Warming Replicas:



There are two pools which maintains the replicas of the partition – **Pool 'A'** and **Pool 'B'**. Read requests are sent to either pool and write requests are sent to both the pools.

In this scenario, initially write request were sent to **Backend 'N'**.When it went down, the read requests were re-directed to **Backend '0'** and write requestswere slowly buffered into **Backend 'N*'** so that the **warming server** slowly warms up and catches up with the **serving replicas**. [6]

References

[1] Blog.twitter.com. (2019). *Introducing Gizzard, a framework for creating distributed datastores*. [online] Available at: https://blog.twitter.com/engineering/en_us/a/2010/introducing-gizzard-a-framework-for-creating-distributed-datastores.html [Accessed 16 Nov. 2019].

[2] Users.umiacs.umd.edu. (2019). [online] Available at: http://users.umiacs.umd.edu/~jimmylin/publications/Leibert_etal_SoCC2011.pdf [Accessed 16 Nov. 2019].

[3] Medium. (2019). *System design for Twitter*. [online] Available at: https://medium.com/@narengowda/system-design-for-twitter-e737284afc95 [Accessed 17 Nov. 2019].

[4] Florian Leibert, Jake Mannix, Jimmy Lin, and Babak Hamadani. *Automatic Management of Partitioned, Replicated Search Services* [online] Available at: https://users.umiacs.umd.edu/~jimmylin/publications/Leibert_etal_SoCC2011.pdf [Accessed 18 Nov. 2019]

[5] Blog.twitter.com. (2019). *Building a complete Tweet index*. [online] Available at: https://blog.twitter.com/engineering/en_us/a/2014/building-a-complete-tweet-index.html [Accessed 17 Nov. 2019].

[6] Slideshare.net (2017). *Nighthawk Distributed catching with Redis* [online] Available at: https://www.slideshare.net/RedisLabs/redisconf17-using-redis-at-scale-twitter [Accessed 18 Nov. 2019]