Design : DW Redis

Use Case : Requirements :

① Functional :

(a) ~~[illegible]~~ get(Key) → get a value corresponding to a Key.

(b) Set(Key, value) → Set value to a Key.

(c) Save → Take a snapshot of ~~records~~ cache.

(d) ~~Restore → Load data from file~~

② Non-Functional

(a) Start, Restart, Stop.

(b) ~~P [illegible] when~~ Load data on restart from a file

(c) Dump data on Stop.

(d) Scalable (~~Sca~~ scales out easily with increase in number of requests)

(e) Highly Performance (fast gets and ~~[illegible]~~ sets)

(f) Highly Available (Service network/disk failures)

(g) ~~[illegible]~~ Data versioning and checkpointing

(h) Crash recovery.

Features and some Estimations :

① High hit rate

② ~~[illegible]~~ minimize Cache miss

③ TTL

④ Giga Bytes → Data store

⑤ 50K to 1M QPS → Query per second

⑥ Latency → ≈ 1ms

⑦ Eviction → LRU

⑧ ~~[illegible]~~

Choice of data structure :

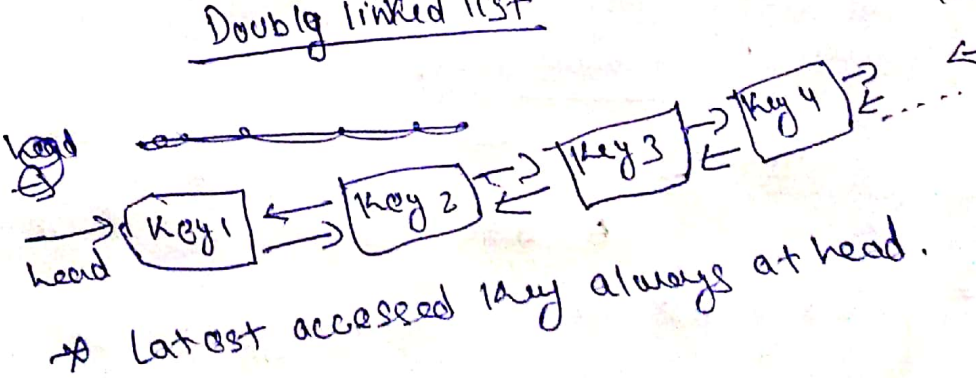
① using Hash Table for key and value store.

② Doubly Linked list for LRU (Eviction).

Data store :

Key 1	(~~Value~~, TTL)
Key 2	(Value, TTL)
Key 3	(Value, TTL)
Key 4	(Value, TTL)
Key 5	(Value, TTL)
Key 6	(Value, TTL)

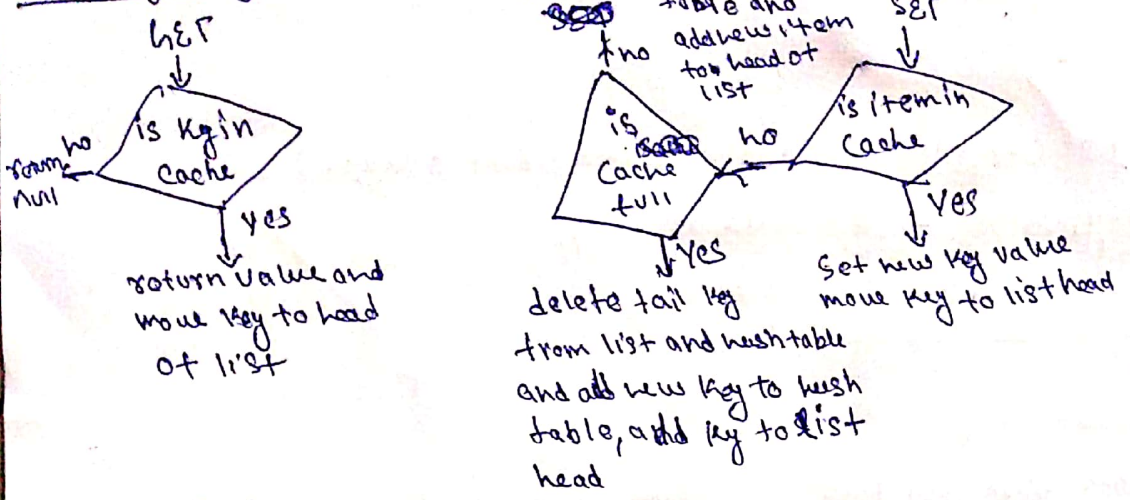
Doubly linked list



* Latest accessed key always at head.

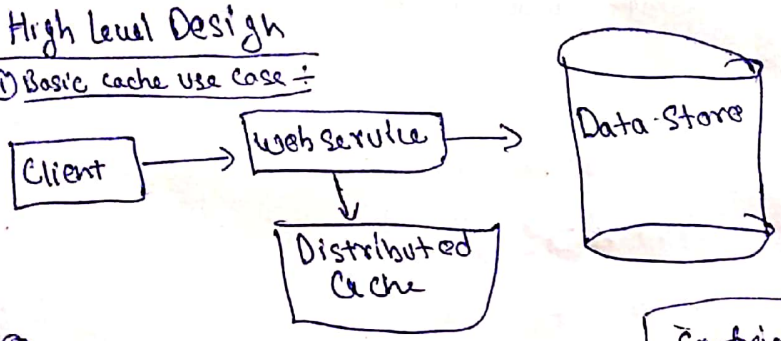
Design : DW Redis

Activity Diagram :

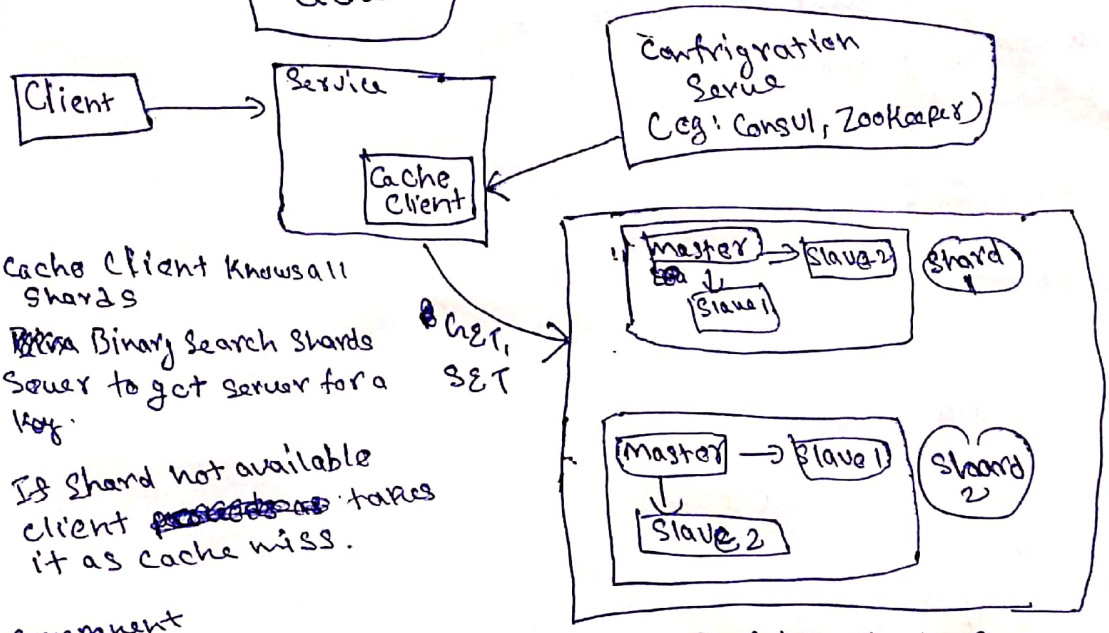


High Level Design

① Basic cache use case :



②

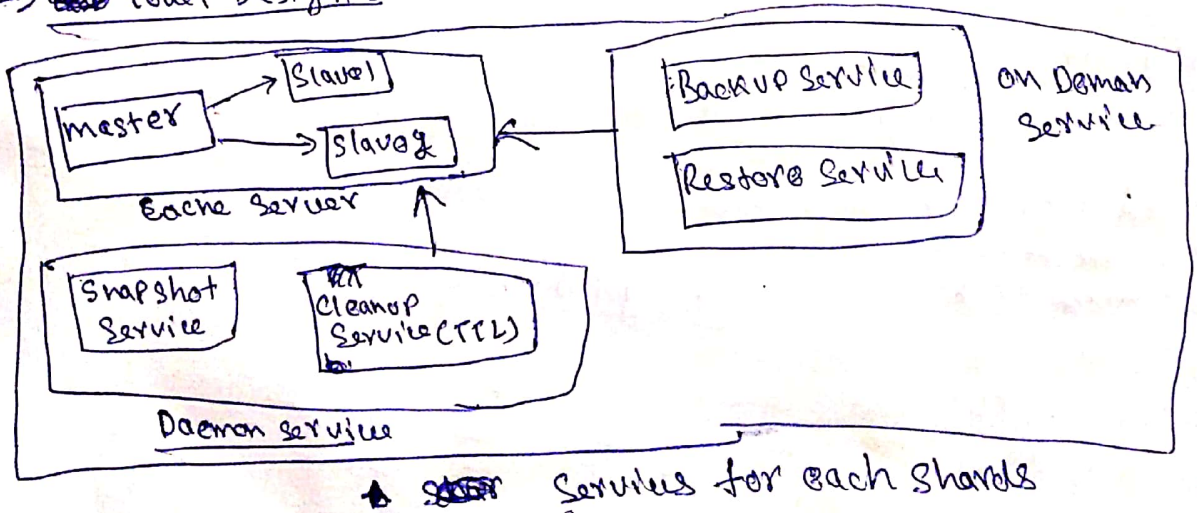


① Cache Client knows all shards

② Binary Search Shards Server to get server for a key.

③ If Shard not available client takes it as cache miss.

Component level Design :



* I have written working code.