* Why MongoDB cannot achieve A in CAP

If a part of a cluster becomes unavailable, a distributed system will either safeguard data consistency by cancelling the request even if it decreases the availability of the system (CP system) or provide availability even though inconsistent data may be returned (AP system).

MongoDB has Consistency and Partition Tolerance; therefore, Availability cannot be guaranteed because if a partition happens between any nodes, the system will have to shut down the inconsistent node until the partition resolves.

* Memcached vs Redis

Both Memcached and Redis:

1. Store data in-memory data stores.
2. Are NoSQL data stores, keeping data as key-value pairs.
3. Open source.

However, there are some differences:

1. Data types: Redis stores data as specific data types so it can change data in place without having to re-upload the entire data value. Memcached stores data only as strings.
   1. Redis supports different data types: String, Lists, Sets, and Hashes.
2. Persistence: Redis allows for persistence to disk, meaning that the data in Redis’s database can be stored and recovered in the event of the Redis server crashing or being rebooted. Memcached does not have the ability to persist to disk natively.
   1. Two ways to persist data in Redis:
      1. AOF Log: AOP stands for Append Only File Log. It works by appending all write operations received by the server. The AOF Log stores all these commands in the same format Redis receives them, which means they can be replayed back to the instance at boot to reconstruct the current state.
      2. RDB Snapshot: RDB stands for Redis Database Backup file. It is a way to take a full snapshot of the current Redis state. The RDB is a compact way to store the current Redis state in a file which can be transferred elsewhere (that is, offsite for disaster recovery).

RDB works by creating a child process of the Redis instance and letting the child process complete the backup.

1. Data length: Redis data keys and strings can be up to 512MB in length. As they are binary safe, you can also store any king of data, even a JPEG image. Memcached supports a key of only 250 bytes and values are capped at 1MB by default.
2. Data Eviction Policies: Redis has multiple ways to deal with data eviction. Memcached can only evict data using Least Recently Used – evicting the data that hasn’t been hit recently. If a data is hit, it is noted.
   1. Ways for Redis to evict data:
      1. No Eviction: just let the memory fill up and then not take any more keys
      2. Volatile TTL (Time to Live): attempt to remove keys with a set TTL first, to preserve data not annotated with a TTL that was meant to persist for longer.
3. Replication: Redis support replication natively and it is able to replicate in a master to follower fashion. Memcached does not support replication without third party software.
4. Clustering: clustering is a way of ensuring high availability of a service by creating a number of instances and connecting them together to form a cluster. Redis offers Redis Cluster.
5. Multithreading: Memcached supports multithreading.

* LRU (146.LRU Cache?)

class LRUCache {

int capacity;

LinkedHashMap<Integer, Integer> cache;

public LRUCache(int capacity) {

if (capacity > 0) {

this.capacity = capacity;

}

this.cache = new LinkedHashMap<>();

}

public int get(int key) {

if (!this.cache.containsKey(key)) { // does not have the key

return -1;

}

setAsRecent(key);

return cache.get(key);

}

public void put(int key, int value) {

if (this.cache.containsKey(key)) { // existed key, we need to update it

this.cache.remove(key);

}

if (this.cache.size() >= this.capacity) { // cache is full

this.cache.remove(this.cache.keySet().iterator().next()); // delete the first one (oldest used)

}

this.cache.put(key, value);

}

private void setAsRecent(int key) {

int value = this.cache.get(key);

this.cache.remove(key);

this.cache.put(key, value);

}

}

/\*\*

\* Your LRUCache object will be instantiated and called as such:

\* LRUCache obj = new LRUCache(capacity);

\* int param\_1 = obj.get(key);

\* obj.put(key,value);

\*/