Functional Requirement
put(key, value)
get(key)

Non-Functional Requirement
Scalable
Highly Available
Highly Performance

Design

Data Structures & Algorithms

- LRU

Components

- **CacheClient**
    - Client knows about all server
    - All cache clients should have same list of cache servers
    - Client stores all list of cache servers in sorted order (e.g TreeMap)
    - Binary Search is used to identify server in O(log n)
    - Client uses **TCP** or UDP protocol to talk to servers
    - If cache host is not available client proceeds as cache miss
- **Maintaining list of cache servers**
    1) Use config management tools like chef & puppet to deploy modified files to every host
    2) Put files in Blob store and let client poll (daemon service) files once a minute or several minutes. Will need to maintain file in storage
    3) Configuration Service to discover cache host and monitor their health.
    4) Asynchronous replication
    5) Split only concrete chard when adding new shar
    6) Master slave replication
- **Configuration Service (Zookeeper)**
    - Monitor leader & follower
    - Failover
    - Leader not working then promote follower to leade
    - Source of authority for clients, cache clients discover cache server from configuration service
    - Is distributed service based nature
    - High Availability
    - Nodes communicate using TCP
    - Zookeeper/Redis Sentinel
- **Consistency**
    - Asynchronous Replication
    - Synchronous replication increases latency
- **Data Expiration**
    - TTL

    - Seperate thread for eviction
- **Security**
    - For internal users only
    - Use firewall
    - Encrypt while reading
- **Monitoring**
  Cache miss, latency, faults, CPU & memory utilization, Network I/O
- **Logging**
  Capture every request in logs, response, emit metrics
- **Consistent Hashing**
    - Domino Effect: if server dies all logs is transferred to next server. This transfer might overload next server then that server could fail causing chain reaction of failures
    - Logs are not spread evenly: Some servers may reside close to each other & some a re apart causing uneven distribution of keys among cache servers
    - Add each server on circle multiple times (Jump Hash Algorithm, Propotional Hash Algorithm)