Day 7: Ghash + Tapestry
Agenda

1. Consistent hash
   Virtual nodes
   Salting

2. Storage systems (Distributed hash)
   Tapestry
   Tables
Cluster LB Goals

- Stateless
- Even utilization
- Consistent map of client to server

- Random function
- Uniform distribution
- Deterministic

 modulo hash:

$server = \text{hash(client IP)} \mod N$

# of servers
During server churn

\[ k = \# \text{ of keys} \]
\[ N = \# \text{ of servers} \]

\[ N = 2 \]
During server churn

$k = \# \text{ of keys}$

$N = \# \text{ of servers}$

$N=2$

$N=3$

adding/delete servers
During server churn

\[ k = \# \text{ of keys} \]

\[ N = \# \text{ of servers} \]

\[ N=2 \]

\[ N=3 \]

\[ k = \frac{k}{N} \] not change
Modular hash: How many keys will move?

original

migration

only \( \frac{k}{n(n+1)} \)
will stay; so
\( \frac{k}{n} \) will stay

\( \frac{k}{n(n+1)} \) will

\[
N \left( \frac{k}{n} - \frac{k}{n(n+1)} \right) = k - \frac{k}{n+1}
\]
What is the problem?

\[ k = \frac{k}{n} \]
Consistent hashing

\[ N = 2 \]

Client ID = \( \text{hash} \text{(IP)} \mod k \)

Server ID = \( \text{hash} \text{(IP)} \mod k \)

KeySpace: \([0 - k - 1] \)
K = 32

\[ \text{server} = [4, 16, 24] \]

Client ID = 12, 13, 10

\[ \text{loadBalance} (\text{client}) \]

\[ \text{for each serverID in server list} \]

\[ \text{if (serverID} < \text{server list}\]
Peer-to-Peer

1. Nodes are owned by random entities
2. Nodes have limited memory/CPU
3. Nodes change membership frequently
func loadBalance (clientIP) {
    ClientID = hash (clientIP) mod N
    ServerList = sort ServerIDs
    for each serverID in ServerList {
        if (serverID > ClientID)
            return serverID
    }
    return ServerList[0]
}

ServerIDs = [3, 35, 56]
N = 64 -> [0-63]
**Consistent hashing**

Motivation: we need to consistently assign clients to servers. Servers maintain client specific state (TCP/session).

Solution: Create server ID & key ID using some equation based on number of servers/clients.

\[ \text{ID} = \left( \text{hash} \left( \text{IP} \right) \mod N \right) \]

Assumptions:
1. Server ID/key ID are evenly distributed.
2. Load balancer maintains a list of all servers.

Diagram:
- DNS
- Clients
- Edge cluster
- Data center
Why can a system have uneven distribution of load?
Why can a system have uneven distribution of load?

1. Server failures (many servers failing)
2. Uneven # of requests from clients
3. Small # of servers which get hashed to similar location
4. Servers with different H/W

\[ S_j = 5 \times \text{CPU of all other server} \]
How do you salt a hash?

Hash \( (h_1 + h_2) \mod N \)

- Give each server \( X \) IPs
- "Salt" the IDs
- Virtual server = give each server "X" IDs
- Get load
- Virtual nodes (solution to uneven distribution)
- "Salt" the IDs
- Get load

4 servers with different HW

2 servers failures (very low)

Why on a solution with uneven distribution of load?
Data Center

Client

Edge Cluster

Data Center

Web Tier

set(k, v)

get(k)

Storage Tier

Distributed hash Table

Key | Value
--- | ---
S1 
S2 
S3
Today

1. Motivation / insight behind consistent hash
2. Consistent hash protocol
   a. Replication
   b. Hardware differences
   c. Uneven server/client
   d. Salting / virtual nodes
3. Distributed hash tables