CSCI 1380

Day 8

Consistent Hash + Tapestry

Last Week

* Consistent hashing

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

Solution: create serverID & keyID (clientID) using some equation

ID = hash(IP) mod N → based # of servers / client
hash → MD5

Assumptions: (1) serverID/keyID are evenly distributed
(2) LoadBalancer maintains a list of all servers

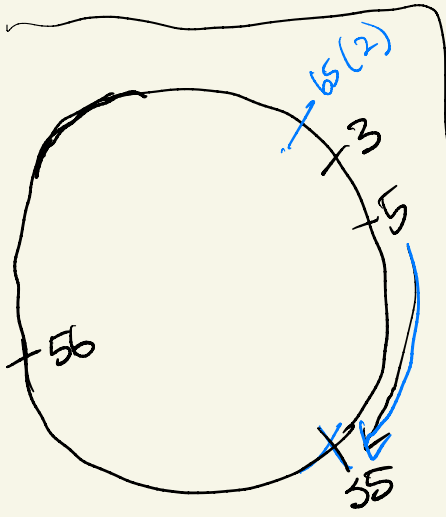
func loadBalance(clientIP)*

clientID = hash(clientIP) mod N

serverlist = sort serverIDs ← need to know all servers

for each serverID in serverlist {
  if (serverID > clientID)
    return serverID
}

return serverlist[0] ← wrap around case



ServerIDs=[3, 35, 56]

N=64 → [0-63]

Why can a system have uneven distribution of load?

1. Server failures (many servers failing) — FB/Amazon failures are temp
* 2. uneven # of requests from clients
* 3. Small # of servers which get hashed to similar location

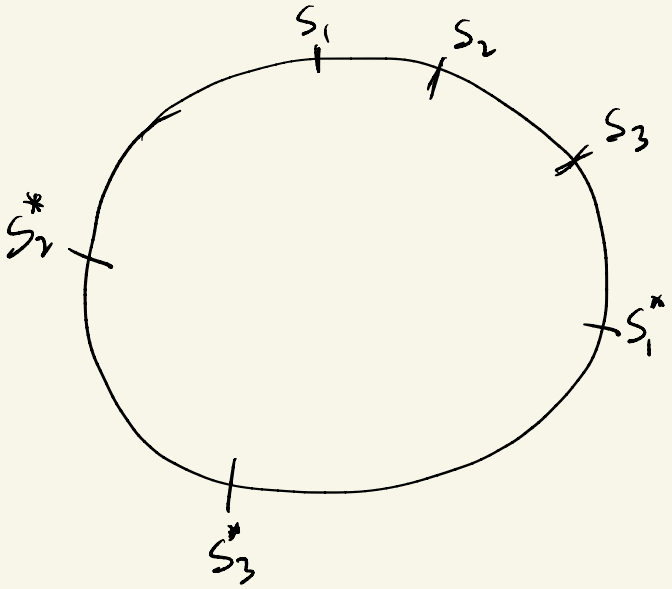
→ Get moar servers

Solution

4. Servers with different H/W

Virtual Server (virtual nodes)

(solution to uneven distribution)



Virtual servers = give each server "X" IDs

1. have X hash functions
* 2. give each server X IPs
* 3. "salting" the IDs

hash("xxx" + IP) mod N

X { ["..."] ["..."] ["..."] ["..."] }

append random string

S_1 ← 10.10.10.100

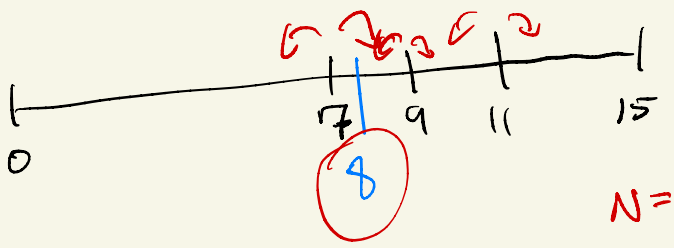
S_1 ← 10.10.10.10

hash(xxx 10.10.10.10) ← salting

How do you set "X"?

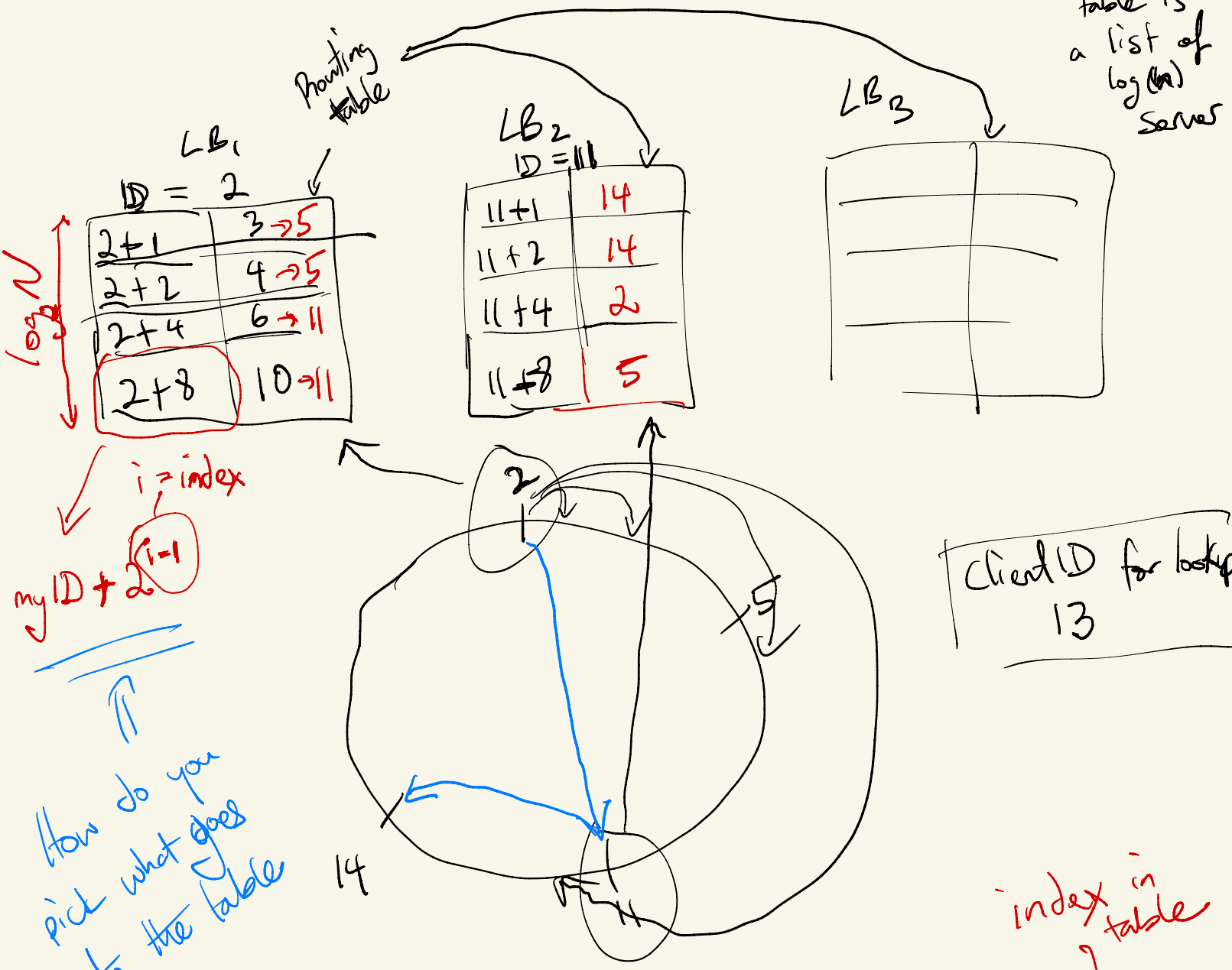
minimize the size of server IDs that each LB needs to maintain?

How does binary search work?



$\log(n)$

$N = 16 \rightarrow \log(16) = 4$



How do you pick what goes into the table

Client ID for lookup
13

lookup for 13 starting at serverID 2

index in table

$i - 1$

Each entry in table is calculated: $myID + 2^{i-1}$

Chord recap

(1) Routing table size $= \log_2(N)$ — N: key space

(2) Each entry in table is $myID + 2^{i-1}$

$myID + 2^{1-1}$	$myID + 1$
$myID + 2^{2-1}$	$myID + 2$
$myID + 2^{3-1}$	$myID + 4$
⋮	

note myID is the server ID of the current server

(3) at each server, you determine the next server (or the server of interest) by checking for the largest ID smaller than the client ID

FB' Cassandra

Amazon's Dynamo

cluster size < 1000 servers

Azure's Service Fabric

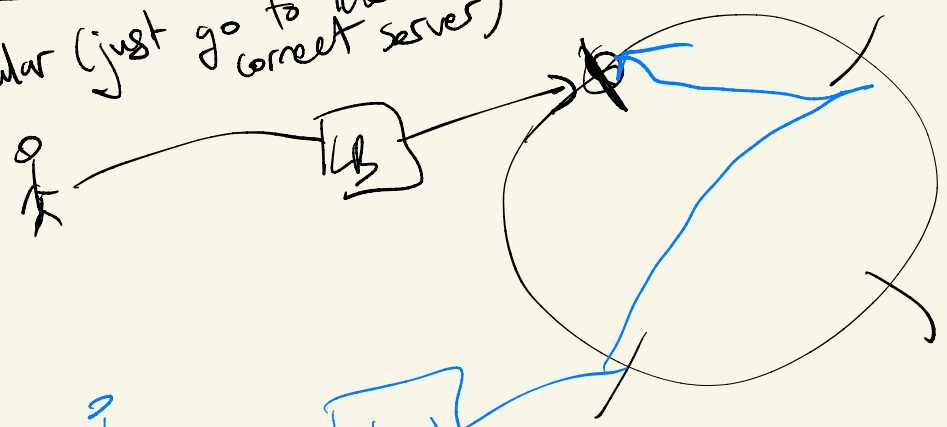
IoT

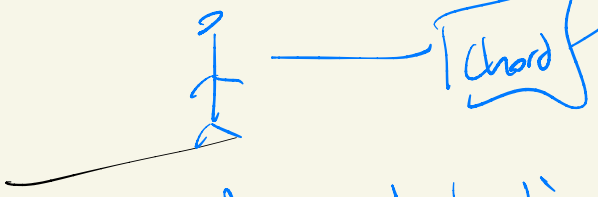
regular consistent hashing

chord version

table size	# of virtual servers	log N
# of servers visited	O(1)	log N
coding level	simple	complex

regular (just go to the correct server)

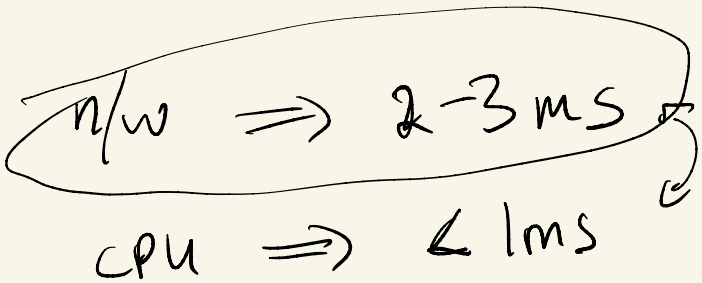




chord - need to binary search through several servers

$$\text{Table size} = \log_2 N = \log_2 64 = 6$$

latencies



n/w latencies are larger than server latencies

thus we analyze big Oh based on # of servers

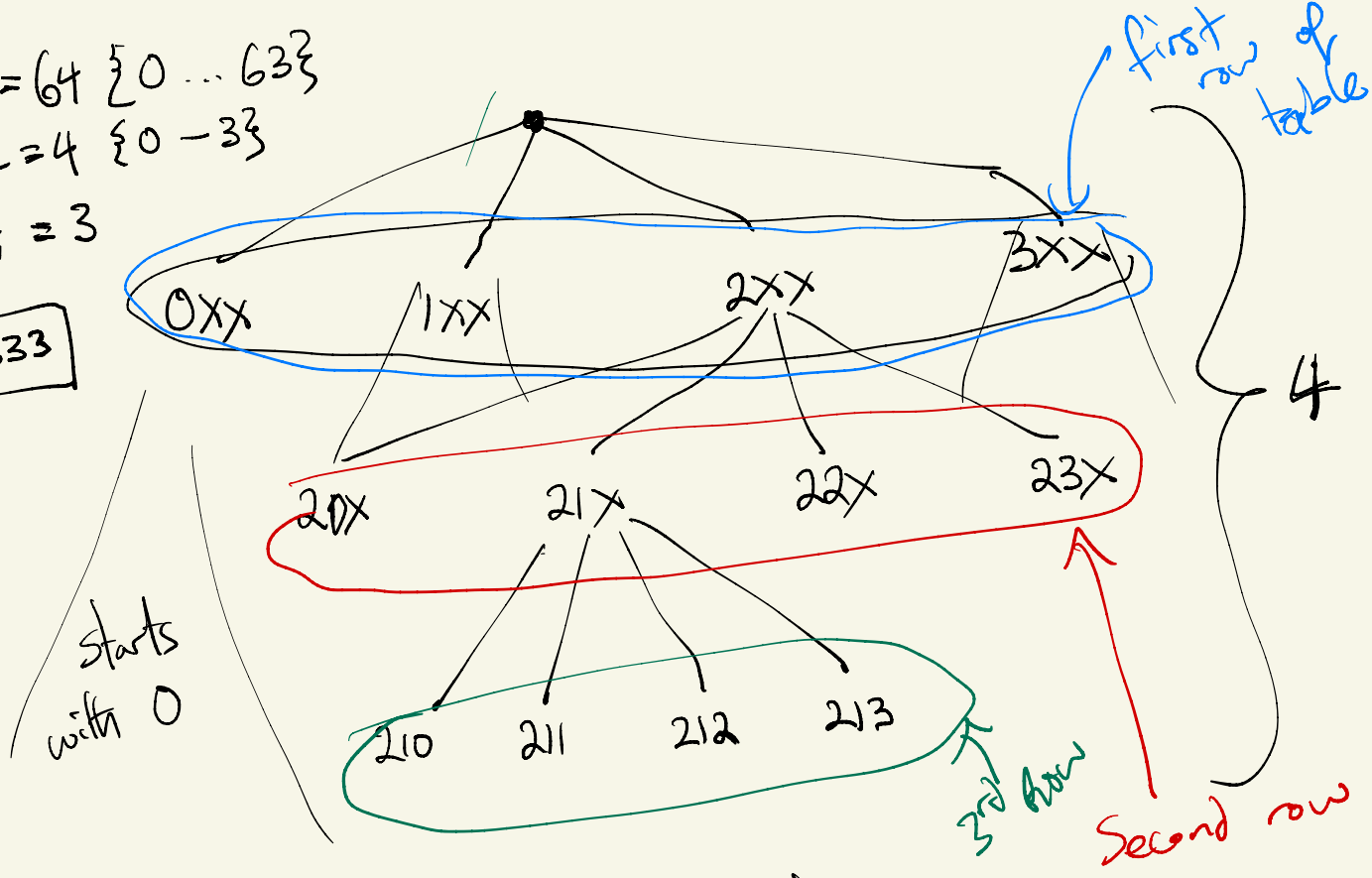
$N = 64$ {0 ... 63}

Base = 4 {0 − 3}

$\log_4 64 = 3$

000 − 333

Base ID space



Key ID Space = 0 ... 63 (N = 64)

↑ base 10

⇒ Pick a Base ⇒ here 4

⇒ 000 − 333

new representation of keyspace with new "Base"

#of cols = Base

3 ← $(\log_B N)$

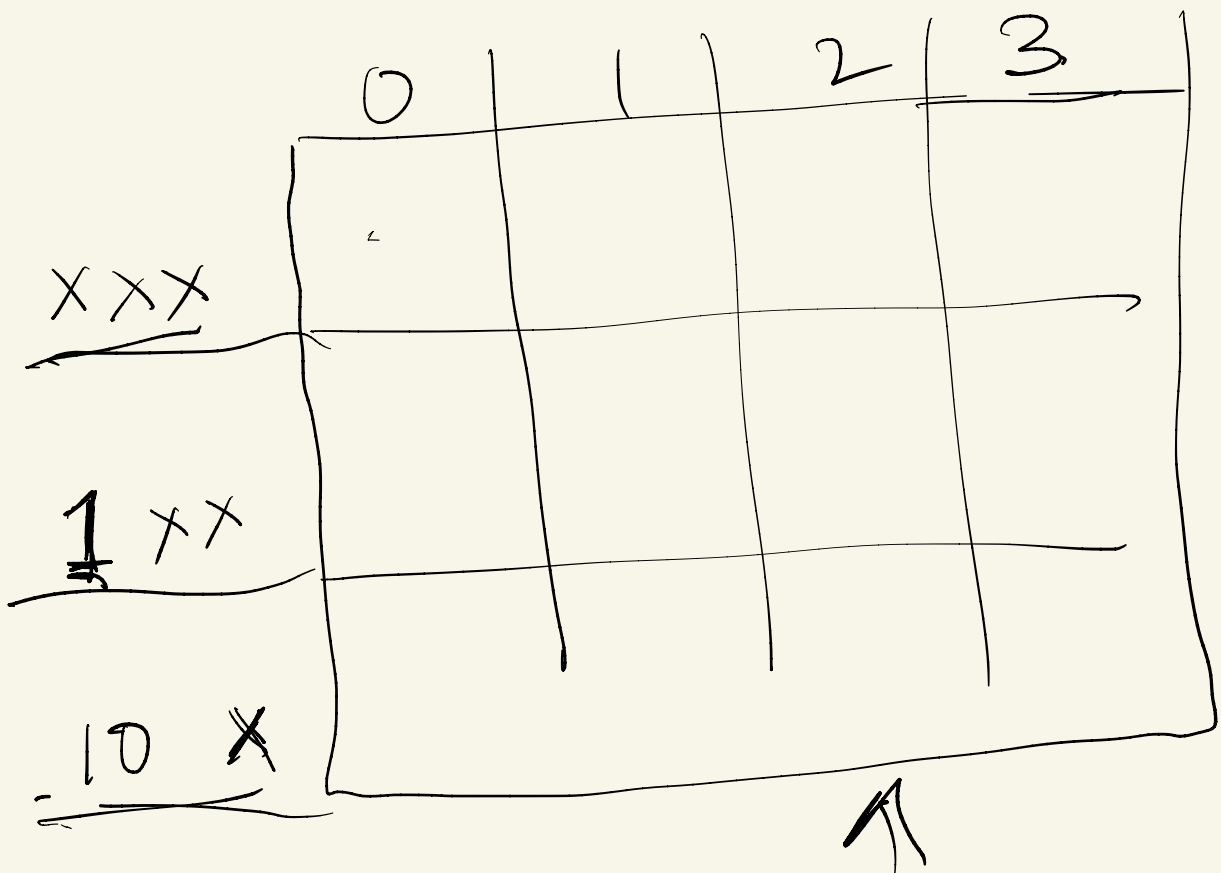
# of rows

0XX	1XX	2XX	3XX
20X	21X	22X	23X
210	211	212	213

← at least one node ID from each top level subtree

Conclude / Summary

(1) assumption in consistent hash
(a) evenly distribution (or lack there of)
(b) Virtual nodes
(c) maintaining a list of server ID
(d) chord (minimize the size of list (routing table))
(e) started a discussion on Tapestry



Server ID → 103

server ID → 321