CSCI 1380
Day 8
Consistent Hash + Tapestry
Last Week

* **Consistent hashing**

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

  Solution: Create server ID & key ID using some equation

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

  Assumptions: (1) Server ID/Key ID are evenly distributed

  (2) LoadBalancer maintains a list of all servers

```python
func loadBalance(clientIP)
    clientID = hash(clientIP) \mod N
    Serverlist = sort serverID
    for each serverID in Serverlist:
        if (serverID > clientID)
            return serverID
    return serverlist[0]
```

ServerIDs = [3, 35, 56]  
N = 64 \rightarrow [0 - 63]
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

Get MOAR servers

Solution

(Virtual nodes)

Virtual server (solution to uneven distribution)

Virtual = give each server "X" IDs

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

Hash \((XXX + IP) \mod N\)

How do you set "X"?
minimize the size of server IDs that each LB needs to maintain?
How does binary search work?

\[ \log(n) \]

\[ N = 16 \rightarrow \log(16) = 4 \]

Client ID for lookup: 13

Index in table = 13 - 1 = 12

lookup for 13 starting at server 12

Each entry in table is calculated: myID + 2
Chord recap

1. Routing table size = \( \log_2 (N) \)

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

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.
### Chord

- **FB Cassandra**
- **Amazon's Dynamo**
- Cluster size < 1000 servers

<table>
<thead>
<tr>
<th>Table Size</th>
<th># of Virtual Servers</th>
<th>log N</th>
</tr>
</thead>
<tbody>
<tr>
<td># of servers visited</td>
<td>O(1)</td>
<td>log N</td>
</tr>
</tbody>
</table>

**Coding Level**
- Simple
- Complex

- Regular (just go to the correct server)

**Chord** - need to binary search through several servers

Table size = \( \log_2 N = \log_2 64 = 6 \)
latencies

\( n/w \Rightarrow 2-3 \text{ms} \)
\( \text{cpu} \Rightarrow \leq 1 \text{ms} \)

\( n/w \) latencies are larger than server latencies

Thus we analyze big Oh based on
\( \# \) of servers
\[ N = 64 \] 0 ... 63
\[ \text{base} = 4 \] 0 to 3
\[ \log_4 64 = 3 \]

Key ID space = 0 ... 63

Pick a base 4

\[ \text{# cols} = \text{base} \]

at least one node ID from each top level sub tree

0 \times x
1 \times x
2 \times x
3 \times x

20 \times
21 \times
22 \times
23 \times

210
211
212
213

first row of table

3rd row

second row

starts with 0

base ID space

000 - 333
Conclude / Summary

0 assumption in consistent hash
(a) evenly distribution (or lack thereof)
(b) virtual nodes
(c) maintaining a list of server ID
(d) chord (minimize the size of list)
(e) started a discussion on Tapestry
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Server ID → 103

Server ID → 321

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XXX

XXX