Day 08: Tapestry
Today

① Imbalance

② Brief overview on Tapestry
A user sends a request to the DNS, which resolves the request to an edge cluster. The edge cluster then directs the request to the load balancer, which distributes the request to various instances of the web tier (front end clusters). The web tier processes the request and returns the results to the user. The request may also need to access the storage tier (back end clusters) for data retrieval or storage.
Imbalance

Hardware diff

give each server a proportional # of keys

small # of servers using virtual nodes & give each x # of keys

small # of clients or small # of popular objects

virtual nodes

Salting
Key = hash(string) % N

& hash function
Distributed hash table (DHT) is a storage that maps consistent hashing keys to servers.
P2P (Peer to Peer)

- Boston
- Japan
- S. Africa

0. H/W <= different resources
2. Nodes come and go at will
3. Geo distributed
P2P = globally + limited CPU/MEM + frequently leave/join

Versus

Industrial storage = within a cluster + lots of CPU/MEM

+ constant # of servers

"quarantine mode"
P2P

need to consider latency

globally distributed

limited cpu/mem

need a smart algo to update the table

versus

Industrial storage

within a cluster

lots of cpu/mem

constant # of servers

server may fail but will eventually return

latency is not a problem

store list of all servers
always publish all your data (every 5 minutes) => ensures that your data stays in tapestry even after failures

Publishing frequency is predefined
Tapestry Node

- Route table
- Back pointers
- Object store

P2P = globally distributed + limited CPU/mem + frequently leave/join (frequent membership)

Need to consider latency

Need an efficient small table

Need a smart data structure
The key space \( W \) = 64

```
[0-3]
```

```
[a b c]
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
0xx
1xx
2xx
3xx
```

```
20x
21x
```

```
210
211
```

```
212
213
```

```
\text{base=4}
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```

```
\text{root = the server/node where a key is stored}
```

```
\text{lookup C}
```

```
212
```
N = 64 \& 0 \ldots 63^2

base = 4 \& 0 - 3^2

\log_{10} 64 = 3

000 - 333

Base ID Space

starts with 0

Key ID Space = 0 \ldots 63

\longrightarrow

pick a

base \Rightarrow 000 - 333

new representation of keyspace

with new "base"

at least one node ID from each top level subtree

#cols = Base

3 \leq \log_B N

# of rows

of tree

first row of table

Second row

3rd row

4
smaller $B = \text{less memory + more next hops}$

bigger $B = \text{more memory + fewer next hops}$

provides controls over tradeoffs

$\log B$ is a measure of some aspect.
Node IDs: 011, 211, 222, 231, 201, 300, 333

\[ N = 64 \]
\[ \text{Base} = 4 \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>011</td>
<td>-</td>
<td>222</td>
<td>333</td>
</tr>
<tr>
<td>xx</td>
<td>201</td>
<td>211</td>
<td>222</td>
<td>231</td>
</tr>
<tr>
<td>23</td>
<td>-</td>
<td>-</td>
<td>231</td>
<td>-</td>
</tr>
</tbody>
</table>

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

1. Uneven distribution
   a. Virtual nodes (hardware differences)
      + uneven server placement

2. Salting (object popularity)

2. Tapestry
   a. Routing
   b. Table creation
   c. Tradeoff
Node IDs: 011, 211, 233, 231, 201, 300, 333

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>211</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>222</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>333</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Lookup (keyID, step)**

- row = step
- col = keyID.at(\text{step})
- nextNode = Table[\text{col}, \text{row}]
- step++
- nextNode.lookup(keyID, step)

**Start @ 0**

- lookup (333, 0) → 211
- lookup (333, 1) → 300
- lookup (333, 2) → 333
Node IDs: 011, 211, 222, 231, 201, 300, 333

```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td></td>
<td>211</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>211</td>
<td>231</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find Start @ 300

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td></td>
<td>211</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
This is a recursive call from node to node where level changes between lookups.

If the lookup happens at a row (say 1) of Node A but there is only one entry in the row, i.e., Node A is the only entry then you have two options:

1. Recursively call Node A & go down a row.
2. Simply go down a row.
Node failure/removal

Graceful

not Graceful
Node failure / removal

Graceful
* Say Good bye to nodes & hand over your data
1. Who to give your data to?
   * New root
2. How to update broken routing tables
   * Need to figure out which tables are impacted by exit!!!
   * Use backpointers to determine nodes to notify

not Graceful
* Failure is detected w/ heartbeats
* Data persists because clients republish the data
  Just disconnect
Better Choice

Versus
closer

my route table
given X potential choices which should I put in table

random note that has me in its route table

during graceful exit: I should provide a replacement

first non overlap

\[ \begin{array}{cc}
XXa & Xxb \\
\end{array} \]

overlap

\[ (a-b) \times b \]

\[ 10ms \]

\[ 5ms \]

\[ 2ms \]

\[ (\text{In Paper}) \]

\[ (\text{Project}) \]