Precept 6
Hashing & Partitioning

Peng Sun

Server Load Balancing
- Balance load across servers
- Normal techniques:
  - Round-robin?

Limitations of Round Robin
- Packets of a single connection spread over several servers

Limitations of Round Robin
- Different RTT on paths
- Packet reordering

Multipath Load Balancing
- Balance load over multiple paths
- Round-robin?

Data Partitioning
- Spread a large amount of data on multiple servers
- Random?
- Very hard to retrieve
Goals in Distributing Traffic

- Deterministic
  - Flow-level consistency
  - Easy to retrieve content from servers
- Low cost
  - Very fast to compute/look up
- Uniform load distribution

Hashing to the Rescue

- Map items in one space into another space in deterministic way

Basic Hash Function

- Modulo
  - Simple for uniform data
  - Data uniformly distributed over $N$, $N >> n$
  - Hash $fn = \langle \text{data} \rangle \mod n$
- What if non-uniform?
  - Typically split data into several blocks
  - e.g., SHA-1 for cryptography

Hashing for Server Load Balancing

- Load Balancing
  - Virtual IP / Dedicated IP Approach
    - One global-facing virtual IP for all servers
    - Hash clients' network info (srcIP/port)
    - Direct Server Return (DSR)

Load Balancing with DSR

- Reverse traffic doesn’t pass LB
- Greater scalability

Equal-Cost Multipath Routing

- Balancing flows over multiple paths
  - Path selection via hashing
    - # of buckets = # of outgoing links
    - Hash network info (src/dst IP) to links
Data Partitioning

- Hashing approach
  - Hash data ID to buckets
  - Data stored on machine for the bucket
  - Cost: \(O(\# \text{ of buckets})\)
- Non-hashing, e.g., “Directory”
  - Data can be stored anywhere
  - Maintenance cost: \(O(\# \text{ of entries})\)

But...

- Basic hashing is not enough
- Map data onto \(k=10\) servers
  - with \((\text{data ID}) \mod k\)
- What if one server is down?
  - Change to \(\mod (k-1)\)?
  - Need to shuffle the data!

Consistent Hashing

- Servers are also in the key space (uniformly)
- Red Nodes: Servers’ positions in the key space
- Blue Nodes: Data’s position in the key space
- Which Red Node to use:
  - Clockwise closest

Another Important Problem

- How to quickly answer YES or NO?
- Is the website malicious?
- Is the data in the cache?

Features of Consistent Hashing

- Smoothness: Addition/removal of bucket does not cause movement among existing buckets (only immediate buckets)
- Spread and load: Small set of buckets that lie near object
- Balanced: No bucket has disproportionate number of objects

Properties We Desire

- Really really quick for YES or NO
- Okay for False Positive
  - Say Yes, but actually No
- Never False Negative
  - Say No, but actually Yes
Bloom Filter

- Membership Test: In or Not In
- \( k \) independent hash functions for each data
- If all \( k \) spots are 1, the item is in.

Demo of Bloom Filter

Start with an \( m \) bit array, filled with 0s.

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

To insert, hash each item \( k \) times. If \( H(x) = a \), set \( \text{Array}[a] = 1 \).

| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

To check if \( y \) is in set, check array at \( H(y) \). All \( k \) values must be 1.

| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

Possible to have a false positive: all \( k \) values are 1, but \( y \) is not in set.

| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

Application of Bloom Filter

- Google Chrome uses BF:
  - First look whether website is malicious
- Storage services (i.e., Apache Cassandra)
  - Use BF to check cache hit/miss
- Lots of other applications...

Thanks!
Hashing in P2P File Sharing

- Two Layers: Ultrapeer and Leaf
- Leaf sends hash table of content to Ultrapeer
- Search request floods Ultrapeer network
- Ultrapeer checks hash table to find leaf

Applying Basic Strategy

- Consider problem of data partition:
  - Given document X, choose one of k servers to store it
- Modulo hashing
  - Place X on server \( i = (X \mod k) \)
  - Problem? Data may not be uniformly distributed
  - Place X on server \( i = (\text{hash}(X) \mod k) \)
  - Problem? What happens if a server fails or joins (\( k \rightarrow k \pm 1 \))?

Use of Hashing

- Equal-Cost Multpath Routing
- Network Load Balancing
- P2P File Sharing
- Data Partitioning in Storage Services