Distributed Systems

Day 4: Distributed Hash Tables [Part 1]
Agenda

• Motivation for Caches and Load Balancing
  • Problem statement
  • Challenges

• Consistent Hashing
  • Ensuring balance
  • Tolerating Failures

• Tapestry
Phantom Menace?

Remember the time 'The Phantom Menace' trailer basically broke the internet?

David Matthews
12/17/15 9:15am
Filed to: NEWS ▼
## History of Consistent Hashing (CHash)

<table>
<thead>
<tr>
<th>CHash invented</th>
<th>Akamai founded</th>
<th>Star Wars</th>
<th>Used in Amazon’s Dynamo DB</th>
</tr>
</thead>
</table>

Forms @ Brown

• Forms submitted to CS-staff
  • Students ‘balanced’ across staff
  • Each staff gets an ‘equivalent’ part of the Alphabet

• What are some potential challenges?
Forms @ Brown: Adding new staff.
Forms @ Brown (e.g., Capstone forms)

• Forms submitted to CS-staff
  • Students `balanced’ across staff
  • Each staff gets an `equivalent’ part of the Alphabet

• What are some potential challenges?
  • Popular content
  • Failure/Node-Additions

• Requirement: a load balancer that ...
  • Accounts for content popularity
  • Minimizes content movement during failures
Caching Background

• Explores redundancy
• Improves performance

Get me trailer!!!

Node A

In cache? Use cache. Else request from web server
Caching Background

- Explores redundancy
- Improves performance

Node A

Node B

In cache? Use cache. Else request from web server

Which Cache will have my content?
Global Scale Caching

• How do you coordinate:
  • Insertion?
  • Lookup?

• Potential options:
  • Maintain a **global DB** of Key->Server Map
  • **Poll** Caches for **Keys** before requesting

• Need a fast, easy algorithm with no overheads
  • E.g., First letters of the URL?
Caching in CS138

- Cache location based on first letter of your name 😊
How do you ensure Balanced Load?

• Dynamically change mappings?

• Randomly assign keys to servers?
Requirements for Distributed Hash Table

• Map Keys to Values.
  • Mapping function should be easy to calculate
  • Function should be independent of # of servers
  • Function should provide nice load balancing properties
Background for CHash

- Map Keys to Values.
  - Mapping function should be easy to calculate
  - Function should be independent of # of servers
  - Function should provide nice load balancing properties

\[
\begin{align*}
<K, V> & \\
<K, V> & \text{KeyID} = \text{Hash (Name)} \mod N \\
<K, V> & \\
<K, V> & \text{ServerID} = \text{Hash (Server IP)} \mod N
\end{align*}
\]
Why Hash Functions?

Key = Hash (Name) Mod N

- Hash maps the “name” into a uniformly random space
  - A “good” hash function, e.g., MD5 or SHA1.
Consistent Hashing

- Both Keys and Servers are hashed
  - Node A → 7fc5...
  - “Ali” → 32ff...

- Use “Mod” to ensure ID space loops
  \[ \text{KeyID} = \text{Hash (Name)} \mod N \]

- Insight: Both Key/ServerIDs go in same space
How do you "Deal" with Failures?

- Ideally: how many objects should be moved?

  K = # of keys
  N = # of servers
  Ideal = K/N
## Hash Server Names

<table>
<thead>
<tr>
<th>Node A</th>
<th>7fc56270e7a70fa81a5935b72eacbe29</th>
<th>7fc5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node B</td>
<td>9d5ed678fe57bcca610140957afab571</td>
<td>9d5e</td>
</tr>
<tr>
<td>Node C</td>
<td>0d61f8370cad1d412f80b84d143e1257</td>
<td>0d61</td>
</tr>
<tr>
<td>Node D</td>
<td>f623e75af30e62bbd73d6df5b50bb7b5</td>
<td>f623</td>
</tr>
</tbody>
</table>
Failures: Adding or Removing Servers

Adding More Server Leads to more even distribution of the space
Failures: Adding or Removing Servers

F-0 0d61 0
1-7 7fc5 3
8-9 9d5e 2
10-F f623 3

F-0 0d61 0
1-7 3a3e 1
4-7 7fc5 2
8-9 9d5e 2
10-F f623 3

F-0 0d61 0
1-7 3a3e 1
4-7 7fc5 2
8 8006 0
9 9d5e 2
10-F f623 3
Consistent hashing: Practicum

- Where are these keys stored?
  - A239
  - 0238
  - 7329
  - 9232
  - 8234
Consistent hashing: Practicum

- Where are these keys stored?
  - A239
  - 0238
  - 7329
  - 9232
  - 8234
Consistent hashing: Practicum

- Adding these servers?
  - 1392
  - d323
  - E234

- Who do they get keys from?
Consistent hashing: Practicum

- Adding these servers?
  - 1392
  - d323
  - E234

- Who do they get keys from?
Still Need More Load Balancing?
Still Need More Load Balancing? How do we Reduce Variance?

• Insight: need more to add more "nodes"
  • Not enough IDs to get statistical properties

• Virtualize nodes
  • Option 1: multiple Chash networks
    • Give keys/Nodes multiple IDs
  • Option 1: give a node multiple names.
    • Virtual copies: make virtual copies of servers
  • Option 2: give an object multiple keys.
    • Salting: make virtual keys
Create multiple Chashs

- Each server now has multiple IDs – one for each Chash
- Each Chash has a different hash algorithm. Why?
Replication: Multiple Chash

• **Insertion:**
  - Calculate all N keys
    - One for each Chash
  - Insert key into each Chash

• **Lookup:**
  - pick a random Chash
    - Chash–determines Hash function
  - Calculate key
  - Lookup key

Create multiple Chash's
• Each server now has multiple IDs – one for each Chash
• Each Chash has a different hash algorithm. Why?
Replication: Re-salting

- **Insert an object multiple times into a** Chash
  - Challenge: Need a way to create multiple keys for an ID
  - Copy object != different key
  - Use salting to create different keys.

- **Insertion:**
  - Calculate all N keys
    - One with each salt
  - Insert each key

- **Lookup:**
  - Pick a random salt (Extra Step)
  - Calculate key
  - Lookup key

Key1 = Hash (Salt0+Name) Mod N
Key2 = Hash (Salt1+Name) Mod N
Key3 = Hash (Salt2+Name) Mod N

Why does this work?
MD5(000000) = 670b14728a....
MD5(0000001) = da292230f0....
Replication: ReSalt V. Multiple Chash

One hash function: slightly different names keys

Key1 = Hash (Salt0+Name) Mod N
Key2 = Hash (Salt1+Name) Mod N
Key3 = Hash (Salt2+Name) Mod N

Key1 = Hash1 (Name) Mod N
Key2 = Hash2 (Name) Mod N
Key3 = Hash3 (Name) Mod N
Replication: ReSalt V. Multiple Chash

- **Lookup:**
  - Pick a random salt
  - Calculate key
  - Lookup key

One hash function: slightly different names keys

Multiple hash functions

- **Lookup:**
  - Pick a random Chash
  - Chash--determines Hash function
  - Calculate key
  - Lookup key
Replication: ReSalt V. Multiple Chash

- **Insertion:**
  - Calculate all N keys
    - One with each salt
  - Insert keys into one Chash

- **Insertion:**
  - Calculate all N keys
    - One for each Chash (different hash function)
  - Insert keys into each Chash