Hashing

Barna Saha
Random Load Balancing

Suppose a content delivery network like YouTube receives a million content requests per minute. Each request needs to be served from one of the 1000 servers. How should one distribute the load so that no server is overloaded.

– Assign each request to a random server.
Random Load Balancing

• Let there be “n” requests and “k” servers

• Consider server “i”

• Define an indicator random variable $X_j$ which will be 1 if request $j$ is assigned to server i and 0 otherwise.

• Load on machine i: 

$$X = \sum_{j=1}^{n} X_j$$

• $E[X] = \sum_{j=1}^{n} E[X_j] = \frac{n}{k}$ [Apply the Chernoff Bound]
Random Load Balancing

- Applying the Chernoff Bound we get

\[
P \! \text{rob} \left( X \geq \frac{n}{k} + 3 \sqrt{\frac{n \ln k}{k}} \right)
\]

\[
= P \! \text{rob} \left( X \geq \frac{n}{k} \left(1 + 3 \sqrt{\frac{\ln k}{n/k}} \right) \right)
\]

\[
\leq e^{-\frac{n}{k} \left( \frac{9 \ln k}{n/k} \right)} = \frac{1}{k^3}
\]
Random Load Balancing

• Apply Union Bound

• Prob(there exists at least one server which is overloaded) $\leq \frac{1}{k^2}$
Balls in Bin

• Suppose you throw m balls into n bins randomly and uniformly, m \(\geq n\).

• If m=n, then the expected number of balls in each bin is just one.

• **EXTRA CREDIT** [20]: Show that the maximum number of balls in any bin is \(O\left(\frac{\ln n}{\ln \ln n}\right)\) with probability \(1 - \frac{1}{n}\).
Throw the balls uniformly at random in the bins
Throw the balls uniformly at random in the bins

Which bin does Ball 7 occupy?
Hash Function

• A hash function from a universe $U=[0,1,2,..,m-1]$ into a range $[0,1,2,..,n-1]$ can be thought of as a way of placing $m$ balls into $n$ bins.

• We have a family of hash functions $H$ and we choose one function from this family uniformly at random.
Perfectly Random Hash Functions

• A family of hash functions $H$ is perfectly random if the following holds

$$\text{Prob}_{h \sim H}(h(x) = y) = \frac{1}{n} \text{ for all } x \in [1, m] \text{ and } y \in [1, n]$$

$$\text{Prob}_{h \sim H}(h(x_1) = y_1 \cap h(x_2) = y_2 \cap \ldots \cap h(x_k) = y_k) = \frac{1}{n^k} \text{ for all } k, x_i \text{ and } y_j$$
Perfectly Random Hash Functions

- A family of hash functions $H$ is perfectly random if the following holds

$$\Pr_{h \sim H}(h(x) = y) = \frac{1}{n} \text{ for all } x \in [1, m] \text{ and } y \in [1, n]$$

$$\Pr_{h \sim H}(h(x_1) = y_1 \cap h(x_2) = y_2 \cap \ldots \cap h(x_k) = y_k) = \frac{1}{n^k} \text{ for all } k, x_i\text{s and } y_j\text{s}$$

Finding hash functions that are perfectly random is difficult in practice. Also, storage and time required to compute such hash functions become prohibitive.
Strongly Universal Hash Family

- Let \( U \) be a universe with \( U = \{0, 1, 2, .., m - 1\} \) and let \( V = \{0, 1, 2, .., n - 1\} \). A family of hash functions \( \mathcal{H} \) is said to be strongly \( k \)-universal if for any element \( x_1, x_2, .., x_k \in U \), any values \( y_1, y_2, .., y_k \in U \), and a hash function \( h \) chosen uniformly at random from \( \mathcal{H} \), we have

\[
\text{Prob}(h(x_1) = y_1 \cap h(x_2) = y_2 \cap ... \cap h(x_k) = y_k) = \frac{1}{n^k}
\]
Example 1 (A 2-strongly Universal Family of Hash Functions). Consider the family of hash functions obtained by choosing a prime number $p \geq m$, letting

$$h_{a,b}(x) = ((ax + b) \mod p) \mod n$$

and then taking the family

$$\mathcal{H} = \{h_{a,b} \mid 1 \leq a \leq p - 1, 0 \leq b \leq p\}$$

Note that it is important that $a$ cannot be 0
Applications

• Password Checker
  – Stores a dictionary of unacceptable password
  – When a user tries to set a password, it is first checked with this dictionary

• Possible solutions
  – Store the passwords in alphabetical order
    • Binary search
  – Use Hash function and store it in a hashtable
Applications

• Password Checker
  – Stores a dictionary of unacceptable password
  – When a user tries to set a password, it is first checked with this dictionary

• Possible solutions
  – Store the passwords in alphabetical order
    • Binary search $O(\log m)$
  – Use Hash function and store it in a hash table
    • $O(1)$ expected time
Applications

• Spam Detection
  – Prevents sending spam emails to the inbox by keeping a dictionary of acceptable email ids.
  – When an email arrives check if it belongs to the list and accept if it does.

• Possible Solutions
  – Store the email ids in alphabetical order
    • Binary search $O(\log m)$
  – Use Hash function and store it in a hash table
    • $O(1)$ expected time
Limitations

• Worst case time could be large
• Space usage may not be ideal
Bloom Filter

• Hash table has size “m” but now it stores only bits
  – Saves space

• Worst case time to search is O(1)
  – Saves time
Bloom Filter

What are the false positive and false negative rates?

Start with an array of 0s.

```
0 0 0 0 0 0 0 0 0 0 0 0
```

Each element of $S$ is hashed $k$ times; each hash gives an array location to set to 1.

```
0 0 1 0 1 1 0 0 1 0 1 0
```

To check if $y$ is in $S$, check the $k$ hash locations. If a 0 appears, $y$ is not in $S$. If only 1s appear, conclude that $y$ is in $S$. This may yield false positives.