CS 591-06, Lecture 12

Jared Saia
University of New Mexico

The Congestion Problem

The WWW suffers from distribution delays. Two causes of delays:

- Congested Networks
- Swamped servers

Servers and networks can be swamped unexpectedly (e.g. “cool site of the day” phenomena)

Solution to Congestion

- Caching is the solution to the congestion problem
- Nearby cache can serve a page quickly even if page source is swamped
- Good for the users and good for the group

Naive Approach

- Naive Approach: Provide a group of users with a single shared cache machine
- Problem: if cache fails, all users are cut off
- Problem: a single cache can be a bottleneck
- Problem: Storage space of a single cache is limited so suffers false misses
- Problem: Limited number of users means limited amount of aggregated requests

Cooperative Caching

- Cooperative Caching: Every client has one *primary cache*. If the request misses there, the client tries to locate the page in other cooperating caches.
- $+$: fault tolerance, scalability and request aggregation
- - : consumes excess bandwidth and requires data duplication

Akamai Approach

- Hash-based scheme - *consistent hashing*
- A user requesting a page contacts only the one cache responsible for that page, no inter-cache communication
- $+$: less communication overhead per page request, discovers misses quickly, reliable
- - (?) : requires mathematical tools to develop the hashing scheme

Hashing

- Goal: Let any client hash a URL to a particular cache containing that URL
- Naive Approach: Hash URL u to cache via typical function like $h(u) = 7u + 4 \mod 23$
- Problem1: What happens when a 24th cache is added to the system? (If we change to new function $h'(u) = 7u + 4 \mod 24$, then essentially every URL is mapped to a different cache.)
- Problem2: What happens if all of the clients have different ideas of which servers are available?

A View

- Each client has a different idea of what servers are available (since information propogates at different speeds)
- A *view* for a particular client is the set of servers which it knows are available
- Because each client has a different view, each client will have a different hash function!

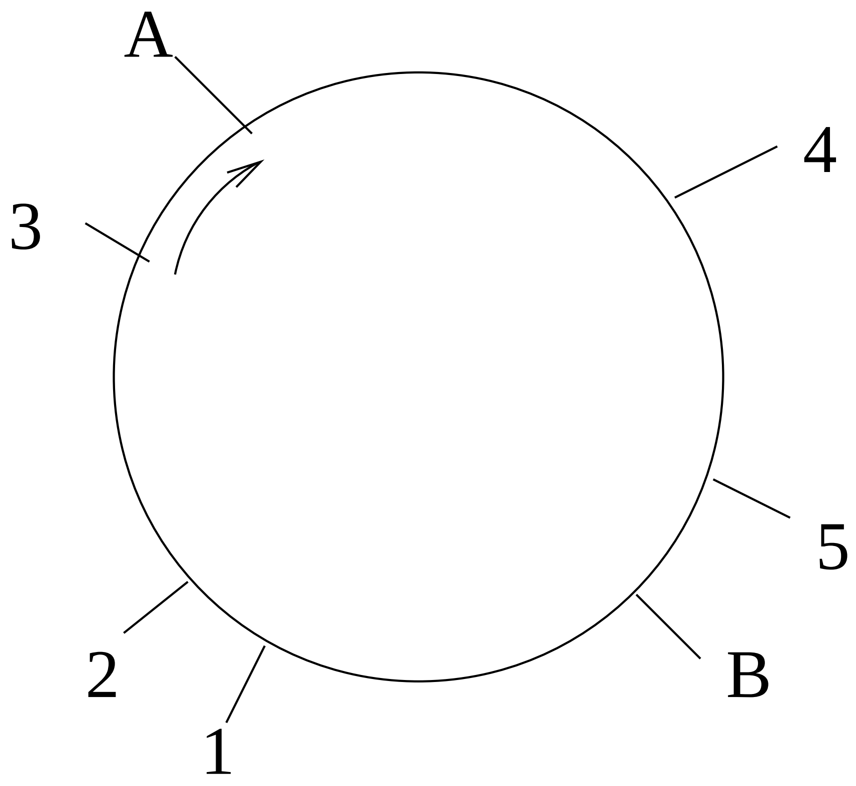
Desirable Properties

- *Monotonicity*: Adding a new cache to any view potentially remaps URLs only from old caches to the new cache. No URLs are remapped from an old cache to another old cache.
- *Balance*: In any view, URL's are distributed uniformly over the caching machines in that view
- *Load*: Over all views, no machine gets too much more than the average number of URL's
- *Spread*: No URL is stored on too many caches

Consistent Hashing

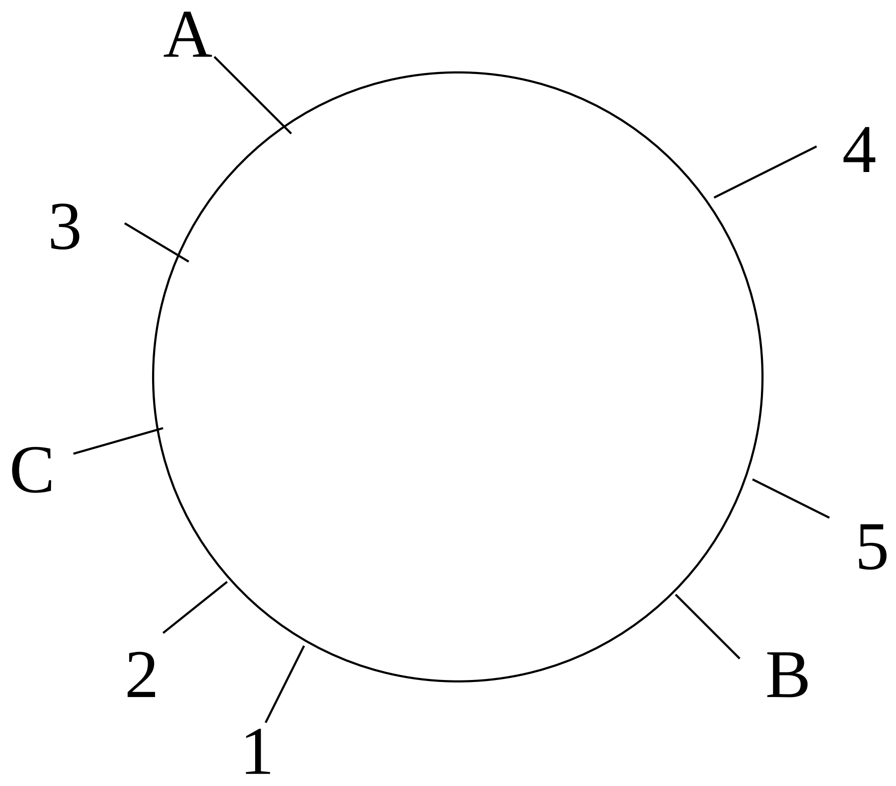
- Choose some standard base hash function that maps strings to the range $[0, \ldots, 1]$.
- This function maps every URL and every cache machine to the unit circle
- Now assign each URL to the first cache whose point it encounters moving clockwise from the URL point

Hashing Example



URL 3 is assigned
to cache A

Cache Insertion Example



When cache C is added, only items 1 and 2 move to the new cache!

Hash Time

- Note that for a given view, we can store all the cache points of that view in a binary tree
- Then when we want to "lookup" a URL, we first find where it falls on the unit circle and then lookup the nearest cache point in the binary tree
- This takes $O(\log C)$ time where C is number of caches. There is a more technical way to do it in $O(1)$ time.

A technical point

- For technical reasons, we need to hash each cache to $O(\log C)$ points on the unit circle where C is the number of cache machines
- Everything else about the hashing algorithm is the same.
- This duplication of cache points ensures a more uniform distribution of pages to caches

Analysis

Intuition:

- *Monotonicity*: When we add cache c to system, it "steals" only those URL points that are near it. These are the only URLs that change who they are hashed to.
- *Balance*: In any view, all cache points are closest to about the same number of URL points
- *Load*: A cache is stuck with an item if it's the closest cache to the item in some view. But if the cache is far from the item, it's likely that in every view, some other cache is closer.
- *Spread*: Since only caches close to an item are likely to be stuck with it in some view, it's likely that only a few caches will ever be responsible for any item

Assumptions

- Let C be the number of caching machines
- Let V be the number of views (and therefor the number of clients). Assume $V = \rho C$ where ρ is some constant.
- Let U be the number of url's, and assume $U = \tau C$ where τ is some constant.
- Assume that each view contains at least $C/2$ cache machines.

Consistent Hashing Properties

Consistent Hashing has the following properties:

- *Monotonicity*: When we add a cache to any view, it "steals" only those URL points that are near it. These are the only URLs that change who they are hashed to.
- *Spread*: No URL is stored in more than $O(\log C)$ caches
- *Load*: Over all views, no machine gets more than $O(\log C)$ times the average number of URL's
- *Balance*: In any view, URL's are distributed uniformly over the caching machines in that view

Initial Observations

- Consider some fixed interval I on the unit circle of length $2k/C$ for some constant k

Questions:

- Q1: For a fixed view, what is the probability that no "cache point" falls in the interval I?
- Q2: What is the probability that in some view, no "cache point" falls in I?

Question 1

- Q1: For a fixed view, what is the probability that no "cache point" falls in the interval I?
- Each "cache point" fails to fall in I independently, with probability $1 - 2k/C$
- There are at least $C/2*(\log C/2)$ such points "cache points".
- Hence the probability of failure is no more than $(1-2k/C)^{C/2*(\log C/2)}$. This probability is no more than $e^{-k \log C/2}$

Question 2

- Q2: What is the probability that in some view, no "cache point" falls in I?
- Let γ be this bad event and let γ_i for $1 \leq i \leq V$ be the event that I contains no cache point in the i-th view
- By the Union bound, we know

$$
\begin{aligned}
P(\gamma) &\leq \sum_{i=1}^{V} P(\gamma_i) && (1)\\
&\leq \sum_{i=1}^{V} e^{-k \log C/2} && (2)\\
&= V * (2/C)^k && (3)\\
&= \rho * (2^k/C^{k-1}) && (4)
\end{aligned}
$$

- For k chosen dependent only on ρ, this probability can be made arbitrarily small!

Useful Lemma

To summarize, we've just proven the following important lemma:

- *Lemma:* For a fixed interval I of size $2k/C$, the event that in some view, no cache point falls in I is very small (i.e. $\rho * (2^k/C^{k-1})$).

Spread

- Consider some fixed URL which is cached to the point p on the unit circle
- Let I be the interval of length $k * 2/C$ for fixed k such that p is the leftmost point of I
- We've shown that in every view, at least one cache point is in I
- Thus, only cache points which fall in I can ever be responsible for caching the given URL
- Q: How many cache points could fall in this interval of size $k * 2/C$?

Spread

- Q: How many cache points could fall in this interval of length $k * 2/C$?
- Let X be the total number of cache points that fall in I
- Each cache point falls in I independently with probability $k * 2/C$
- By linearity of expectation, $E(X) = (C \log C) * k * 2/C = 2k * \log C = O(\log C)$
- By Chernoff bounds, the probability that X deviates from $E(X)$ by much is very small
- By a union bound, the probability that *any* URL contains more than $O(\log C)$ cache points in its interval is small

Chernoff Bounds Review

- Let $X = \sum_{i=1}^{m} X_i$ where the X_i are random variables which are: independent and all take on values between 0 and 1.

then for any $0 < \delta \leq 1$,

- $P(|X - E(X)| \geq \delta E(X)) \leq 2e^{\frac{-E(X)\delta^2}{4}}$

The Details

- For all $1 \leq j \leq U$, let I_j be the interval of length $k * 2/C$ whose rightmost point is the same as the point that the i-th URL is hashed to on the unit circle.
- For any $1 \leq j \leq U$, let X_j be the total number of cache points that fall in I_j
- For any $1 \leq j \leq U$, we know that $E(X_j) = (C \log C) * k * 2/C$
- By Chernoff Bounds, $P(|X_j - E(X_j)| \geq (2k * \log C)) \leq 2e^{\frac{-2k \log C}{4}}$

Details

- For any $1 \leq j \leq U$, let γ_j be the event that I_j has more than $4k \log C$ cache points in it. Then:

$$
\begin{aligned}
P(\bigcup_{j=1}^{U} \gamma_i) &\leq \sum_{i=1}^{U} P(\gamma_i) && (5)\\
&\leq \sum_{i=1}^{U} 2e^{\frac{-2k \log C}{4}} && (6)\\
&= \tau C * 2e^{\frac{-2k \log C}{4}} && (7)\\
&= e^{\frac{-2k \log C + 4 \log(2\tau C)}{4}} && (8)
\end{aligned}
$$

Load

- Load Fact: Over all views, no cache machine gets more than $O(\log C)$ times the average number of URL's
- To show this, note that each cache machine maps to $O(\log C)$ points on the unit circle
- We can say it is responsible for all the URL points that map to intervals of length $k * 2/C$ associated with these points
- Finally note that we don't expect too many of the URL points to fall in any of these intervals!
- Again we use LOE, Chernoff bounds, and union bounds.