Consistent Hashing

Last Class: Web Caching

- If you want to access a webpage lot of times then it is quite efficient to cache it in your disk

- If you request *amazon.com*, and you also requested it in the recent past, then the page can be served from your local cache. If not, you incur a cache miss, and the page is downloaded and stored in your local cache
  - The most obvious benefit is that the end user experiences a much faster response time
  - **But it also improves the internet:** Less web traffic, less-congestion, less communication, less dropped packet.

We can take caching to the Next Level

- Web cache that is shared by many users
- **Idea**: What if all users on Rice Network (or close in location) can access amazon.com from the some web cache located nearby?
- **The benefits**: By aggregating the recent page requests of a large number of users, these users will enjoy many more cache hits and consequently less latency.
- The business plan for **Akamai Technology** in 1998 which is currently valuated at more than **10 billion**.

Challenges

- Remembering the recently accessed Web pages of a large number of users might take a lot of fast storage, with efficient retrieval.
  - Akamai's plan was to do this in main memory.

- Implementing a shared cache at a large scale requires spreading the cache over multiple machines

- Suppose there are 100 machines and you are requesting amazon.com, which machine should you go it?

Some ideas

- Poll all 100 servers for copy.
  - Infeasible at scale.

- **Our training so far**: Use hash functions and for say *amazon.com*, let h(*amazon.com*) machine cache the page associated with amazon.com.
  - Easy to search!

- What if you got 5 extra machines tomorrow? What if a server with #ID, crashed and it down.
  - It turns out this will be quite frequent at large scale.

Data Center at Google

“In each cluster's first year, it's typical that 1,000 individual machine failures will occur; thousands of hard drive failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; 5 racks will "go wonky," with half their network packets missing in action; and the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span, Dean said. And there's about a 50 percent chance that the cluster will overheat, taking down most of the servers in less than 5 minutes and taking 1 to 2 days to recover.”

Quote from Jeff Dean https://www.datacenterknowledge.com/archives/2008/05/30/failure-rates-in-google-data-centers

Reallocation?

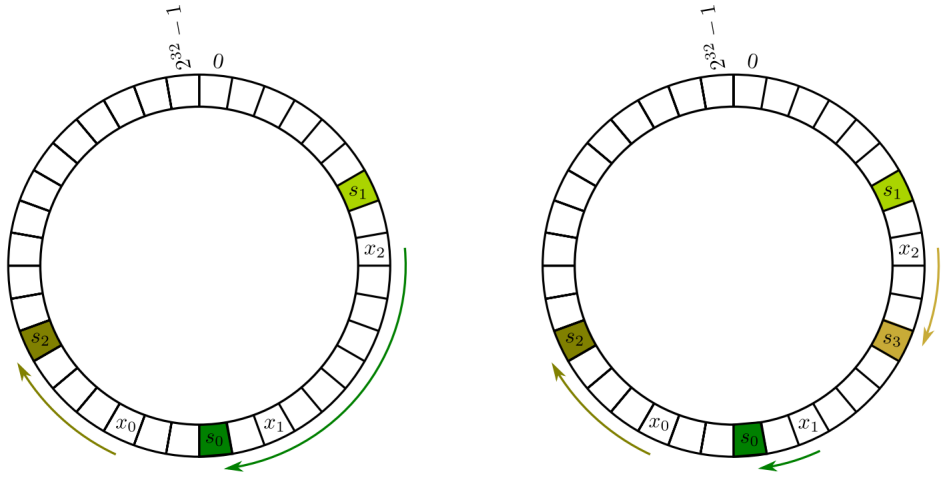
- h(x) = x mod 12
  - Addition or deletion of one machine changes it to x mod 13 or x mod 11. Shift everything to maintain consistency.
  - Infeasible when n is changing all the time.

- Thoughts?

- Solution: Consistent Hashing

So what is consistent hashing.

- Hash both machines and objects in the same range.



- To assign object x, compute h_i(x) and then traverse right until you find the first machine's hash $h_m(Y)$. Assign x to Y.

Image taken from Tim Roughgarden Notes

What happens when servers are added/deleted?

- They all share the load.
- We only have to move data from one of the servers instead of all with traditional hashing!

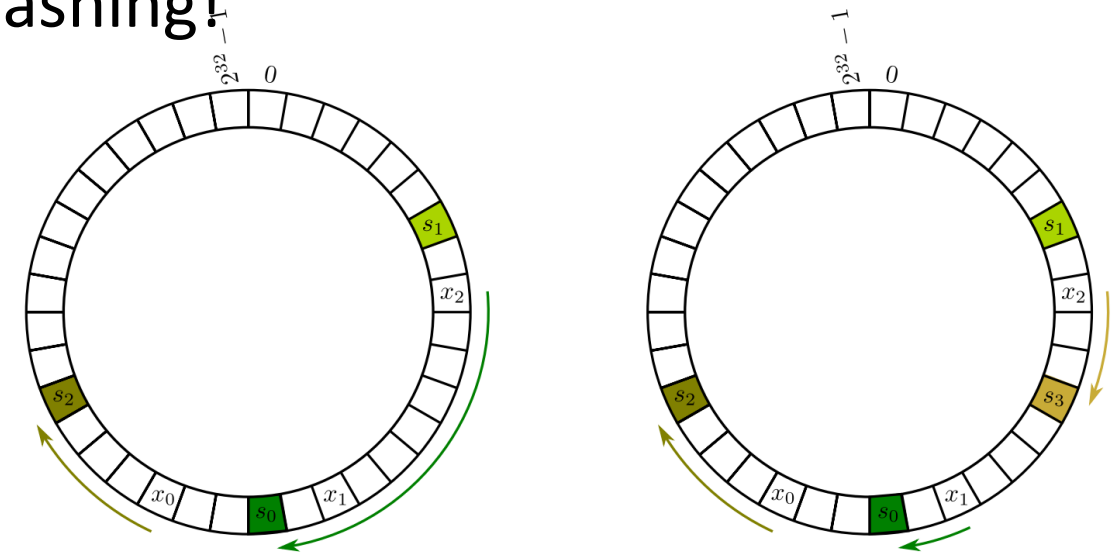


Image taken from Tim Roughgarden Notes

Search time?

- A fraction of the array will be scanned.

- Can we do something smarter?

- Can do log(n)
  - Use Binary Search trees.
  - Put allocated index of serves in a binary search tree. Update as needed.
  - Given h(x) find its successor in log(n) time.

Pseudocode using Binary Search Tree

- Insert an item x:
  - Find the successor of h_i(x) in the BST (if it has no successor in the BST then return the machine with the smallest hm value)
  - Store x in the returned machine.

- Insert a new machine Y :
  - Find the successor of h_m (Y) in the BST (if it has no successor in the BST then return the machine with the smallest hm value)
  - Move all items whose h_i value is less than h_m(Y) to the newly inserted machine Y .

- Delete an item x:
  - Find the successor of h_i (x) in the BST (if it has no successor in the BST then return the machine with the smallest hm value)
  - Delete x in the returned machine

- Delete an existing machine Y :
  - Find the successor of h_m(Y) in the BST (if it has no successor in the BST then return the machine with the smallest hm value)
  - Move all items in Y to the returned machine

Some History

- 1997: The implementation of consistent hashing given in this lecture first appeared in a research paper in STOC ("Symposium on the Theory of Computing") — this is one of the main conferences in theoretical computer science. Ironically, the paper had previously been rejected from a theoretical computer science conference because at least one reviewer felt that "it had no hope of being practical."

- 1998: Akamai is founded.

- March 31, 1999: A trailer for "Star Wars: The Phantom Menace" is released online, with Apple the exclusive official distributor. apple.com goes down almost immediately due to the overwhelming number of download requests. For a good part of the day, the only place to watch (an unauthorized copy?) of the trailer is via Akamai's Web caches. This put Akamai on the map.

Contd..

- April 1, 1999: Steve Jobs, having noticed Akamai's performance the day before, calls Akamai's President Paul Sagan to talk. Sagan hangs up on Jobs, thinking it's an April Fool's prank by one of the co-founders, Danny Lewin or Tom Leighton.

- September 11, 2001: Tragically, co-founder Danny Lewin is killed aboard the first airplane that crashes into the World Trade Center. (Akamai remains highly relevant to this day, however.)

- 2001: Consistent hashing is re-purposed to address technical challenges that arise in peer-to-peer (P2P) networks. A key issue in P2P networks is how to keep track of where to look for a file, such as an mp3.
  - This functionality is often called a "distributed hash table (DHT)." DHTs were a very hot topic of research in the early years of the 21st century.

Contd..

- 2006: Amazon implements its internal Dynamo system using consistent hashing. The goal of this system is to store tons of stuff using commodity hardware while maintaining a very fast response time. As much data as possible is stored in main memory, and consistent hashing is used to keep track of what's where.

- This idea is now widely copied in modern lightweight alternatives to traditional databases (the latter of which tend to reside on disk).

Some Math

- Given m, items and n machines. What is the expected load of each machine?
  - m/n ?
  - Symmetry argument. Equal probability.

- When a machine is added, the expected number of items that move to the newly added machine is $\frac{m}{n+1}$
  - Again straightforward argument.

- With high probability, no machine owns more than O($\frac{logn}{n}$) fraction

Proof

- Fix some interval I with length $\frac{2\log n}{n}$.
- No machine lands in this interval has probability
- $\left(1-\frac{2\log n}{n}\right)^n \approx e^{-2\log n} = \frac{1}{n^2}$

- Split the range into $\frac{n}{2\log n}$ equal sized disjoing intervals
- Probability that there exist one interval where no machine lands is given by the union bound
  - $\leq \frac{n}{2\log n} \times \frac{1}{n^2} \leq 1/n$
- So with probability $\geq 1-\frac{1}{n}$ every interval contains at least one machine.

- So how much does it own (the load) with high probability?
  - $2\times interval\ size = \frac{4\log n}{n}$

How about the other way round

- With high probability we can say no machine is overloaded

- Can we say the same that no machine will be underloaded?

Birthday paradox

- With mere 23 people in room, what is the chance that there exist two people with exactly same birthdays.
  - More than ½ for 365 days!!
  - Compute $\frac{365}{364} \times \frac{364}{363} \times \ldots \times \frac{342}{341} \approx 0.49$ (probability that no two people have same bithdays)

- Split interval into equally n^2 parts, with each taking a fraction $\frac{1}{n^2}$.
  - Birthday paradox says that with high probability two machines will fall in the same bin!!

Can we reduce the variance of workloads in consistent hashing

- With high probability max load scales like O($\frac{\log n}{n}$)
- Create multiple copies of machines and hash
- Why it will reduce variance?

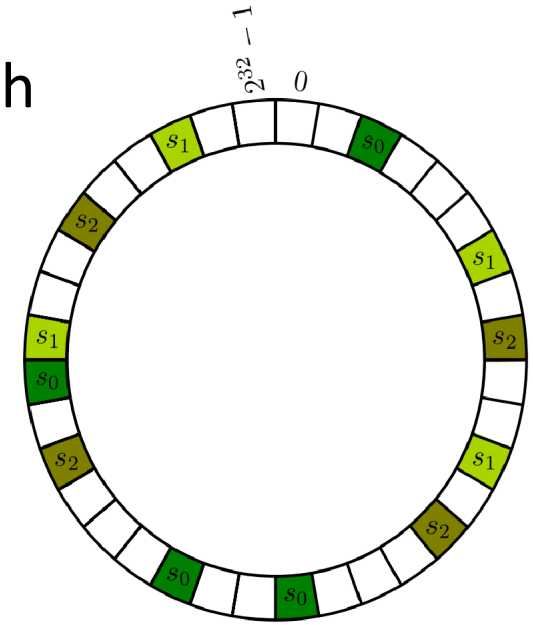


Image taken from Tim Roughgarden Notes

Continued ..

- If we create K copies then the total load is the sum of K i.i.d random variable
  - The sum of i.i.d random variable is sharply (exponentially with K) concentrated around mean. (Chernoff bounds)

Image taken from Tim Roughgarden Notes