COMP 480 Lecture Notes: Consistent Hashing

Scribed by Quan Tran (qmt1), Murad Imre (mi15), Camden Graham (cgg5)

September 23, 2025

1 Bloom Filters and Web Caching

1.1 Motivation for Caching

When a webpage is requested multiple times, it is efficient to cache it.

- Example: If you request `amazon.com` and did so recently, the page can be loaded from your local cache.
- Otherwise, you incur a **cache miss**, requiring a download from the remote server.

1.2 Benefits

- Faster user experience: cached requests load immediately.
- System-level improvements: reduced network traffic, less congestion, fewer communication overheads, fewer dropped packets.

1.3 Rule of Thumb

If something is used more than once, cache it. This heuristic mirrors how humans optimize repeated tasks.

1.4 Examples

- Word usage: A few high-frequency words ("head" of the distribution) account for most traffic. Handling 80–90% of traffic is possible with caching.
- Netflix: Trending movies are cached near users, loading quickly. Rare movies are slower because they may not be cached.
- Geography: Requests routed to faraway caches (e.g., Australia) perform poorly compared to nearby caches.

1.5 Shared Caches

- Idea: A shared cache for multiple nearby users (e.g., all Rice students).
- Benefits: More aggregate hits, less latency.

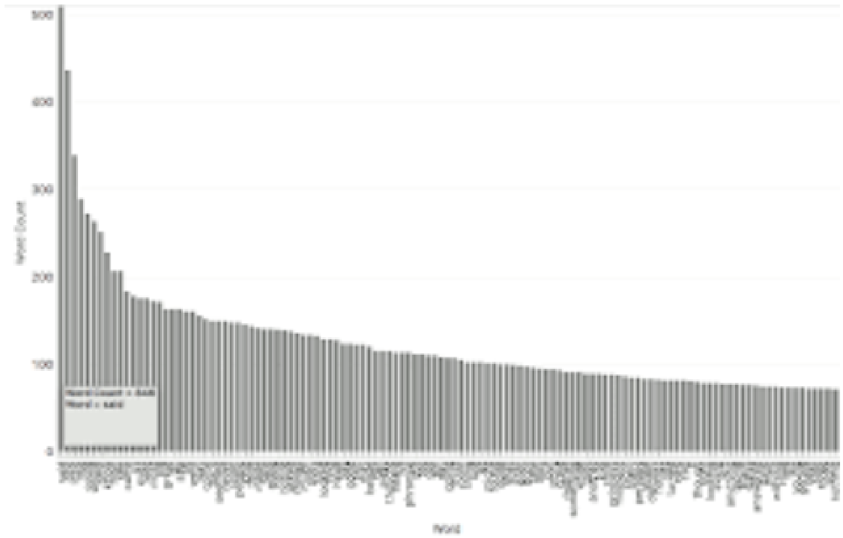

Figure 1: Word frequency distribution illustrating cache efficiency.

- Example: Akamai Technologies (founded 1998, valued at \$10B+) built a CDN business around this idea.

1.6 Challenges

- Storing recently accessed pages for many users requires massive, fast memory.
- Akamai's plan: perform this in main memory for speed.
- At large scale, caches must be distributed over many machines.
- Question: If there are 100 cache servers, which one should handle `amazon.com`?

2 Naïve Solutions and Their Limits

2.1 Initial Ideas

- Poll all servers to check if they hold a cached copy — infeasible.
- Use hashing: map `amazon.com` to server $h($`amazon.com`$)$.

2.2 Problem: Dynamic Changes

- If machines are added/removed frequently, modulo hashing ($h(x) = x \mod n$) forces nearly all keys to remap.
- Example: Switching from $\bmod 12$ to $\bmod 13$ shifts almost everything.

2.3 Machine Failures at Scale

Jeff Dean's observations of Google data centers:

> "In each cluster's first year, ∼1000 machines fail; thousands of disks fail; power units crash; 20 racks go offline; overheating can shut down most servers within 5 minutes."

This highlights the importance of robustness.

3 Consistent Hashing

3.1 Definition

- Hash both servers and objects into the same numeric space, visualized as a **ring**.
- Rule: Assign object x to the nearest server in the clockwise direction from $h(x)$.

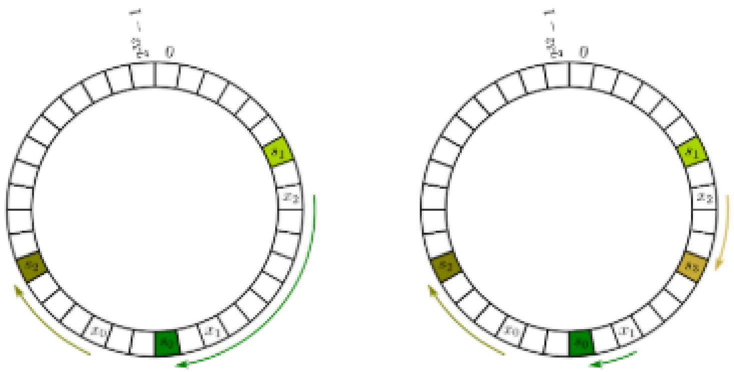

Figure 2: Consistent hashing ring showing server placement and object assignment.

3.2 Advantages

- Adding a server only requires moving the data that maps to it.
- Removing a server reroutes its objects to the next clockwise server.
- Only a small fraction of objects move, unlike in standard modulo hashing.

3.3 Search Time

- Naïve: scanning clockwise — $O(n)$.
- Improved: Store server hashes in a balanced binary search tree; lookup becomes $O(\log n)$.

3.4 Historical Timeline

- 1997: First paper on consistent hashing presented at STOC. Originally rejected for being "impractical."
- 1998: Akamai founded.
- 1999: Star Wars trailer crashes Apple's site; Akamai's CDN handles the demand.

- April 1, 1999: Steve Jobs calls Akamai's President Paul Sagan. Sagan thinks it's an April Fool's prank and hangs up.
- September 11, 2001: Co-founder Danny Lewin tragically killed on the first plane that hit the World Trade Center.
- 2001: Repurposed for peer-to-peer (P2P) networks — foundation of distributed hash tables (DHTs).
- 2006: Adopted by Amazon Dynamo for scalable, memory-based storage on commodity hardware.

4 Mathematical Analysis of Load Balancing

4.1 Expected Load

- Given m items and n machines, the expected load of each machine is $\frac{m}{n}$.
  - This follows by symmetry: each machine is equally likely to be assigned an item.
- When a machine is added, the expected number of items that move to the new machine is $\frac{m}{n+1}$.
  - Again, this is a straightforward argument by symmetry.
- With high probability, no machine owns more than $O\left(\frac{\log n}{n}\right)$ fraction of the total items.

4.2 Proof (from lecture)

- With high probability, no machine is severely **overloaded**.
- Question: Can we also guarantee that no machine is **underloaded**?
- Analogy: The **birthday paradox** illustrates how random allocation can still produce surprising gaps or collisions, raising the underloaded concern.

Overload bound (interval method):

- Fix an interval I of length $\frac{2\log n}{n}$.
- Probability that no machine lands in this interval is:

$$\left(1 - \frac{2\log n}{n}\right)^n \approx e^{-2\log n} = \frac{1}{n^2}.$$

- Divide the unit circle into $\frac{n}{2\log n}$ disjoint intervals.
- By the union bound, the probability that *any* interval is empty is at most:

$$\frac{n}{2\log n} \cdot \frac{1}{n^2} \leq \frac{1}{n}.$$

- Therefore, with probability at least $1 - \frac{1}{n}$, every interval has at least one machine.

- Thus each machine controls at most $\frac{4 \log n}{n}$ fraction of the load, with high probability.

Variance reduction (replication):

- To balance underloaded cases, create K **virtual copies** of each machine.
- The load on each machine is then modeled as the sum of K i.i.d. random variables.
- By standard concentration bounds, the sum is sharply concentrated around its mean.
- This ensures workloads are balanced across machines with high probability.