CS 7280/4973 – Data Str & Alg Scalable Comp Spring 2025

Lecture 22 — April 2, 2025

Prof. Prashant Pandey *Scribe: Rohan Chotirmall*

1 Overview

This lecture introduces distributed systems as the primary motivation for a new hashing technique, *consistent hashing*. It covers the issues that arise when dealing with a distributed system and how consistent hashing addresses these issues.

2 Distributed Systems

We will define a distributed system as a system of m distributed machines. In such a system two major issues arise: load-balancing and scalability.

Problem 1: Load Balancing

We have n objects such that each of them need to be stored in one of the m distributed machines. The goal is to distribute the objects as evenly as possible, so that we can utilize the full power of the system.

Problem 2: Scalability

If the number of machines m changes, objects will need to be distributed to new locations. Unlike in previous cases, where the main bottleneck was memory accesses, in a distributed system the major bottleneck becomes the network. The goal is to minimize the amount of data being transferred between machines in the network (as seen in Figure 1), given that the number of machines will continually change.

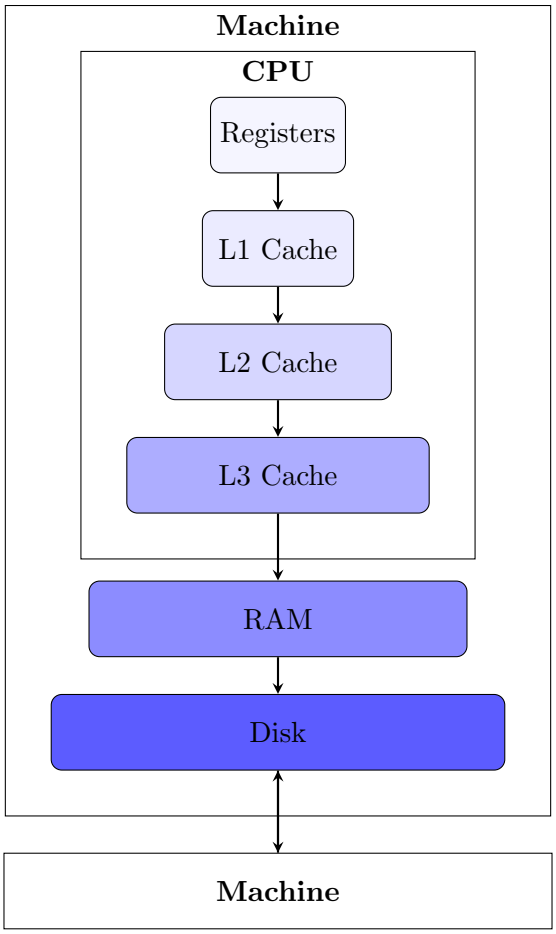

Figure 1: Memory Hierarchy

3 Initial Solution: Hashing

3.1 Hashing Review

Universal Hashing

We are given a **family** H of hash functions mapping from some universe $U \to \{0, \ldots, m-1\}$, such that $\forall x \neq y \in U : |\{h \in H : h(x) = h(y)\}| \leq \frac{|H|}{m}$.

i.e., the probability that x and y collide is $\frac{1}{m}$ if we choose h randomly from H.

Example: $h(x) = [((ax) \mod p) \mod m]$, $0 < a < p$, p is prime.

2-wise Independent Hashing

A **family** H of hash functions mapping from some universe $U \to T, |T| = m$, for which:

$Pr_{h \in H}[h(x_1) = t_1, h(x_2) = t_2] = O(\frac{1}{m^2})$, $x_1 \neq x_2 \in U$

Example: $h(x) = [((ax + b) \mod p) \mod m]$, $0 < a < p, 0 \leq b < p$, p is prime.

3.2 Application to Distributed System

If the number of machines m equals the number of objects n in our distributed system, if we hash each object to determine the node to store it in, the number of objects in the fullest node would be $O(\frac{\log n}{\log \log n})$ WHP, as shown in previous lectures.

We can resolve this problem using a 2-wise independent family of hash functions to create a perfect hashing, thus resolving load-balancing. We can hash each object to one of the m machines.

However, perfect hashing only works well if the number of machines m does not change during the process.

For example, if we add an additional machine to the system, we must either:

1. Change m in h to $m + 1$ to get a new h'.
    - By doing so, we need to move almost all items to their new location.
    - Since the network is the main bottleneck in a distributed system, this will be an expensive undertaking, thus reducing scalability.
2. Keep m unchanged and thus do not move any objects.
    - The new machine will not be used, creating a load imbalance.

We need a strategy that does not incur too much re-hashing while also keeping the load of all machines almost balanced.

4 Consistent Hashing

4.1 Overview

- Each machine is mapped to a random real number in the interval $[0, 1]$ by the hash function h_m
- Each object is mapped to a random real number in the interval $[0, 1]$ by the hash function h_o
- Each object will be stored in the first machine on its right. If no machine is on its right, then store the object in the machine with the smallest hash value.

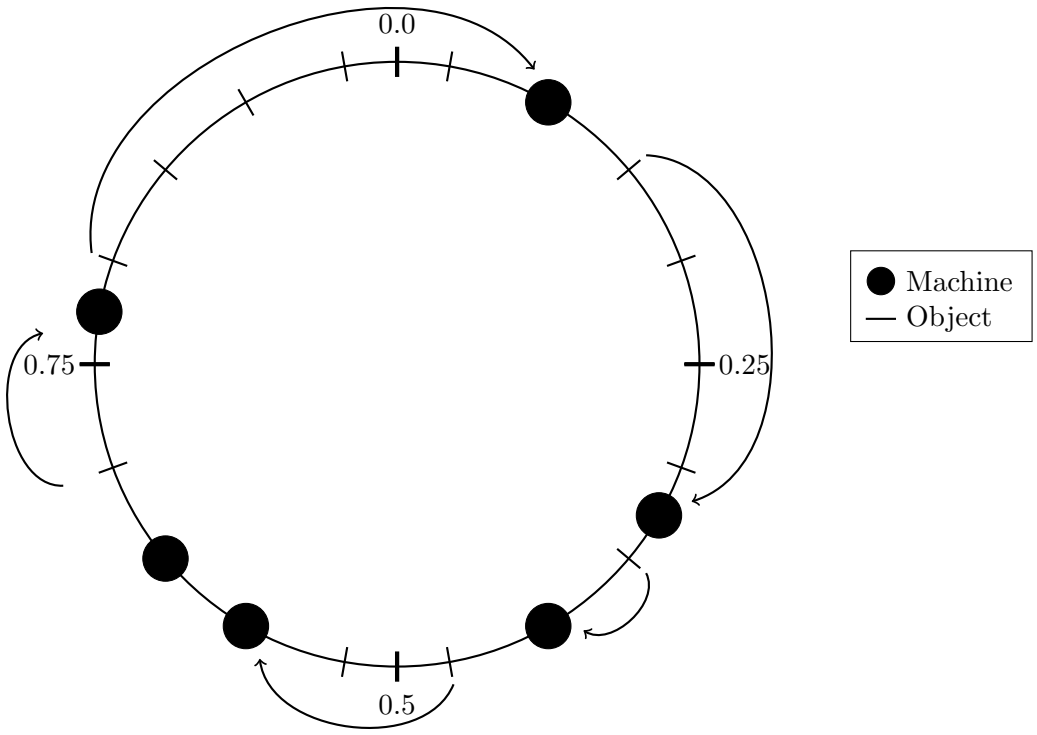

Figure 2: Consistent Hashing Visualization

4.2 Implementation

To dynamically maintain machines and objects, we need to maintain a Binary Search Tree (BST), whose keys are the values assigned to the machines by h_m.

To insert an item x:

- Find the successor of $h_o(x)$ in the BST
- If there is no successor, then find the smallest h_m value in the BST
- Store x in the returned machine

To delete an item x:

- Find the successor of $h_o(x)$ in the BST
- If there is no successor, then find the smallest h_m value in the BST
- Delete x in the returned machine

To insert a new machine y:

- Find the successor of $h_m(y)$ in the BST
- If there is no successor, then find the smallest h_m value in the BST
- Move all items from the returned machine with h_o values less than $h_m(y)$ to the newly inserted machine y

To delete an existing machine y:

- Find the successor of $h_m(y)$ in the BST
- If there is no successor, then find the smallest h_m value in the BST
- Move all items in y to the returned machine

4.3 Bounds

Given n objects and m machines:

Lemma 1: With high probability, no machine owns more than $O(\frac{\log m}{m})$ objects.

Lemma 2: With high probability, the size of the smallest interval assigned to a machine is $O(\frac{1}{m^2})$.

Lemma 3: When a machine is added, the expected number of items that need to move to the newly added machine is $\frac{n}{m+1}$.

Proof of Lemma 2

Fix some interval I of length $\frac{2\log m}{m}$.
Pr[no machine lands in I] $= (1 - \frac{2\log m}{m})^m = ((1 - \frac{2\log m}{m})^{\frac{m}{2\log m}})^{2\log m} \approx \frac{1}{m^2}$
Equally split $[0, 1]$ to $\frac{m}{2\log m}$ such intervals.
By union bounds: Pr[every one of these intervals contains at least 1 machine] $=$
$1 - \frac{m}{2\log m} * \frac{1}{m^2} > 1 - \frac{1}{m}$
With high probability, each machine owns an interval of length at most $\frac{4\log m}{m}$.

References

[1] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, Daniel Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. *STOC*, 654-663, 1997.