COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2026.

Lecture 3

Logistics

- Remember to sign up for Gradescope. Needed to see your in-class quiz grades.
- Second Canvas quiz will be posted after class. Second in-class quiz will be next Monday.

Content Overview

Last Class:

- Application of linearity of expectation to analyzing expected number of collisions in a hash table.
- Markov's inequality: the most fundamental concentration bound.

$$\Pr(X \geq t) \leq \frac{\mathbb{E}[X]}{t}.$$

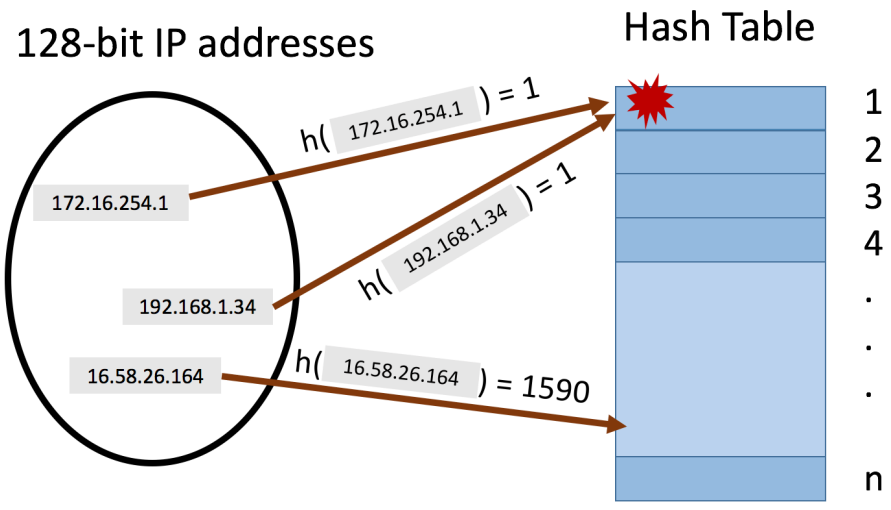
Content Overview

Today:

- Application of Markov's inequality to collision free hashing.
- 2-level hashing for optimal lookup time and storage.
- 2-universal and pairwise independent hash functions.
- Possibly start on application of random hashing to distributed load balancing.
- Through this application learn about Chebyshev's inequality, which strengthens Markov's inequality.
- Connection to the law of large numbers.

Hash Tables

We store m items from a large universe in a hash table with n positions.



- Want to show that when $\mathbf{h} : U \to [n]$ is a fully random hash function, query time is $O(1)$ with good probability.
- Equivalently: want to show that there are few collisions between hashed items.

Collision Free Hashing

Application: Collision Free Hashing

Via linearity of expectation, we shows that the expected number of pairwise collisions is:

$$\mathbb{E}[\mathsf{C}] = \frac{m(m-1)}{2n}.$$

- Say we have a lot of space. In particular, let $n = 4m^2$. Then: $\mathbb{E}[\mathsf{C}] = \frac{m(m-1)}{8m^2} \leq \frac{1}{8}$.
- **Think-Pair-Share:** What is an upper bound on the probability that we have any collisions, i.e., $\Pr[\mathsf{C} \geq 1]$?

Apply Markov's Inequality: $\Pr[\mathsf{C} \geq 1] \leq \frac{\mathbb{E}[\mathsf{C}]}{1} = \frac{1}{8}$.

So with probability at least 7/8 we have no collisions at all and worst-case $O(1)$ query time.

Very fast...but we are using $O(m^2)$ space to store m items...

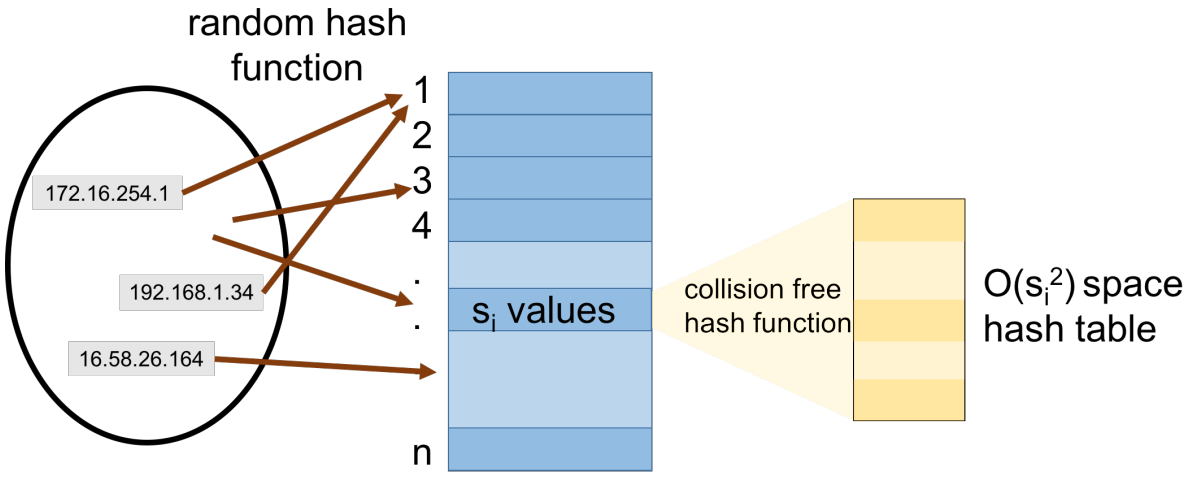
m: total number of stored items, n: hash table size, C: total pairwise collisions.

Two-Level Hashing

Two Level Hashing

Want to preserve $O(1)$ query time while using $O(m)$ space.

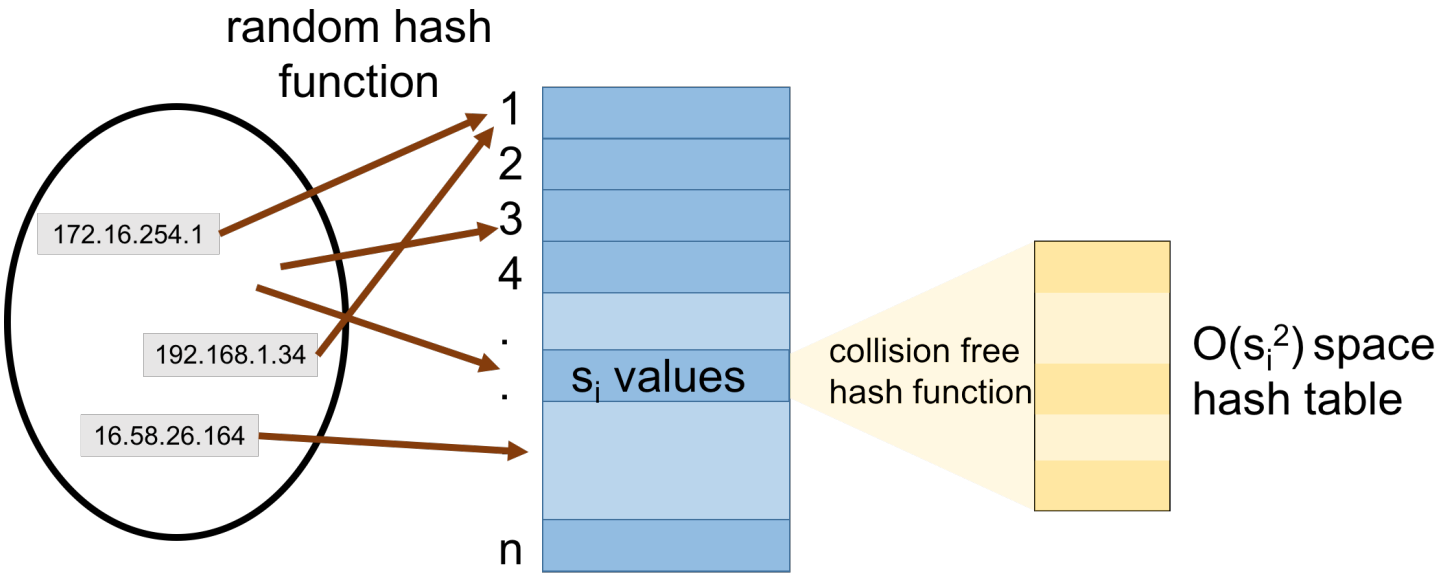
Two-Level Hashing:



- For each bucket with s_i values, pick a collision free hash function mapping $[s_i] \to [s_i^2]$.
- **Just Showed:** A random function is collision free with probability $\geq \frac{7}{8}$ so can just generate a random hash function and check if it is collision free.

Space Usage

Query time for two level hashing is $O(1)$: requires evaluating two hash functions. What is the expected space usage?



Up to constants, space used is: $S = n + \sum_{i=1}^{n} s_i^2 \mathbb{E}[S] = n + \sum_{i=1}^{n} \mathbb{E}[s_i^2]$

$$\mathbb{E}[s_i^2] = \mathbb{E}\left[\left(\sum_{j=1}^{m} \mathbb{I}_{h(x_j)=i}\right)^2\right]$$

$$= \mathbb{E}\left[\sum \mathbb{I}_{\cdot} \cdot \mathbb{I}_{\cdot}\right] = \sum \mathbb{E}\left[\mathbb{I}_{\cdot} \cdot \mathbb{I}_{\cdot}\right]$$

Space Usage

$$\mathbb{E}[\mathbf{s}_i^2] = \sum_{j,k \in [m]} \mathbb{E}\left[\mathbb{I}_{\mathbf{h}(x_j)=i} \cdot \mathbb{I}_{\mathbf{h}(x_k)=i}\right]$$

$$= m \cdot \frac{1}{n} + 2 \cdot \binom{m}{2} \cdot \frac{1}{n^2}$$

$$= \frac{m}{n} + \frac{m(m-1)}{n^2} \leq 2 \text{ if we set } n = m.$$

- For $j = k$, $\mathbb{E}\left[\mathbb{I}_{\mathbf{h}(x_j)=i} \cdot \mathbb{I}_{\mathbf{h}(x_k)=i}\right] = \frac{1}{n}$.
- For $j \neq k$, $\mathbb{E}\left[\mathbb{I}_{\mathbf{h}(x_j)=i} \cdot \mathbb{I}_{\mathbf{h}(x_k)=i}\right] = \frac{1}{n^2}$.

Total Expected Space Usage: (if we set $n = m$)

$$\mathbb{E}[\mathbf{S}] = n + \sum_{i=1}^{n} \mathbb{E}[\mathbf{s}_i^2] \leq n + n \cdot 2 = 3n = 3m.$$

Near optimal space with $O(1)$ query time!

x_j, x_k: stored items, m: # stored items, n: hash table size, $\mathbf{h}$: random hash function, $\mathbf{S}$: space usage of two level hashing, $\mathbf{s}_i$: # items stored at pos i.

Efficiently Computable Hash Functions

Beyond Idealized Hash Function

So Far: we have assumed a **fully random hash function** $\mathbf{h}(x)$ with $\Pr[\mathbf{h}(x) = i] = \frac{1}{n}$ for $i \in 1, \ldots, n$ and $\mathbf{h}(x), \mathbf{h}(y)$ independent for $x \neq y$.

- To compute a random hash function we have to store a table of x values and their hash values. Would take at least $O(m)$ space and $O(m)$ query time to look up $\mathbf{h}(x)$ if we hash m values. Making our whole quest for $O(1)$ query time pointless!

x	h(x)
x_1	45
x_2	1004
x_3	10
⋮	⋮
x_m	12

Efficiently Computable Hash Functions

What properties did we use of the randomly chosen hash function?

Pairwise Independent Hash Function. A random hash function from $\mathbf{h} : U \to [n]$ is pairwise independent if for all $i, j \in [n]$:

$$\Pr[\mathbf{h}(x) = i \cap \mathbf{h}(y) = j] = \frac{1}{n^2}.$$

Exercise 1: Check the two-level hashing proof to confirm that this property is all that was needed.

When $\mathbf{h}(x)$ and $\mathbf{h}(y)$ are chosen independently at random from $[n]$, $\Pr[\mathbf{h}(x) = i \cap \mathbf{h}(y) = j] = \frac{1}{n^2}$ (so a fully random hash function is pairwise independent).

Efficient Implementation: Let p be a prime with $p \geq |U|$. Choose random $\mathbf{a}, \mathbf{b} \in [p]$ with $\mathbf{a} \neq 0$. Represent x as an integer and let

$$\mathbf{h}(x) = (\mathbf{a}x + \mathbf{b} \mod p) \mod n.$$

Universal Hashing

Another common requirement for a hash function:

2-Universal Hash Function (low collision probability). A random hash function from $\mathbf{h} : U \to [n]$ is two universal if:

$$\Pr[\mathbf{h}(x) = \mathbf{h}(y)] \leq \frac{1}{n}.$$

Think-Pair-Shair: Which is a more stringent requirement?
2-universal or pairwise independent?

Pairwise Independent Hash Function. A random hash function from $\mathbf{h} : U \to [n]$ is pairwise independent if for all $i, j \in [n]$:

$$\Pr[\mathbf{h}(x) = i \cap \mathbf{h}(y) = j] = \frac{1}{n^2}.$$

Questions on Hash Tables?

Next Step

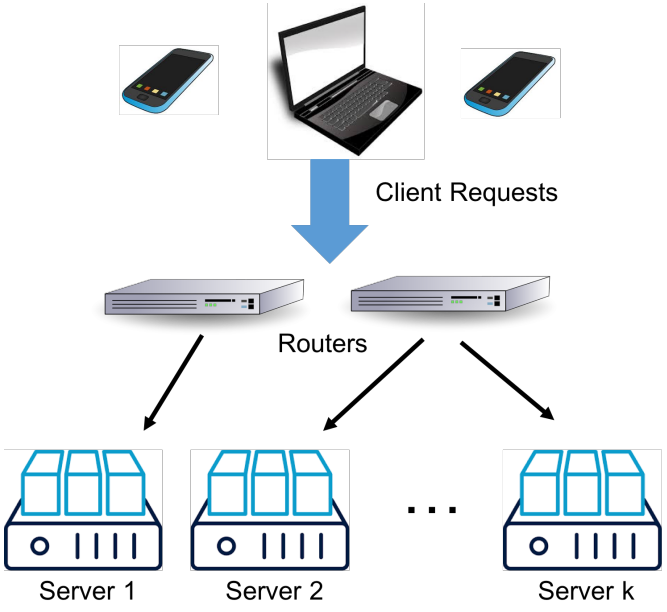
1. We'll consider an application where our toolkit of linearity of expectation + Markov's inequality doesn't give much.

2. Then we'll show how a simple twist on Markov's (Chebyshev's inequality) can give a much stronger result.

Hashing for Load Balancing

Another Application

Randomized Load Balancing:



Simple Model: *n* requests randomly assigned to *k* servers. How many requests must each server handle?

- Often assignment is done via a random hash function. Why?

Weakness of Markov's

$$\mathbb{E}[\mathsf{R}_i] = \sum_{j=1}^{n} \mathbb{E}[\mathbb{I}_{\text{request } j \text{ assigned to } i}] = \sum_{j=1}^{n} \Pr[j \text{ assigned to } i] = \frac{n}{k}.$$

If we provision each server be able to handle twice the expected load, what is the probability that a server is overloaded?

Applying Markov's Inequality

$$\Pr[\mathsf{R}_i \geq 2\mathbb{E}[\mathsf{R}_i]] \leq \frac{\mathbb{E}[\mathsf{R}_i]}{2\mathbb{E}[\mathsf{R}_i]} = \frac{1}{2}.$$

Not great...half the servers may be overloaded.

n: total number of requests, k: number of servers randomly assigned requests, R_i: number of requests assigned to server i.

Chebyshev's inequality

With a very simple twist, Markov's inequality can be made much more powerful.

For any random variable X and any value $t > 0$:

$$\Pr(|X| \geq t) = \Pr(X^2 \geq t^2).$$

X^2 is a nonnegative random variable. So can apply Markov's inequality:

Chebyshev's inequality:

$$\Pr(|X - \mathbb{E}[X]| \geq t) = \Pr(X^2 \geq t^2) \leq \frac{\mathbb{E}[X^2]}{t^2} \frac{\mathrm{Var}[X]}{t^2}.$$

(by plugging in the random variable $X - \mathbb{E}[X]$)