CS 591, Lecture 4
Data Analytics: Theory and Applications
Boston University

Charalampos E. Tsourakakis

February 1st, 2017

Recap Lecture 3

n balls into n bins

Last time we saw that:

$$\mathbf{Pr}\left[\exists \text{bin with more than } \frac{3\log n}{\log\log n} \text{ balls}\right] \leq \frac{1}{n}$$

We also saw that by changing the maximum load

$$c\frac{\log n}{\log\log n}$$

by playing with constant c, we can decrease the failure probability as $\frac{1}{\text{poly}(n)}$.

n balls into n bins

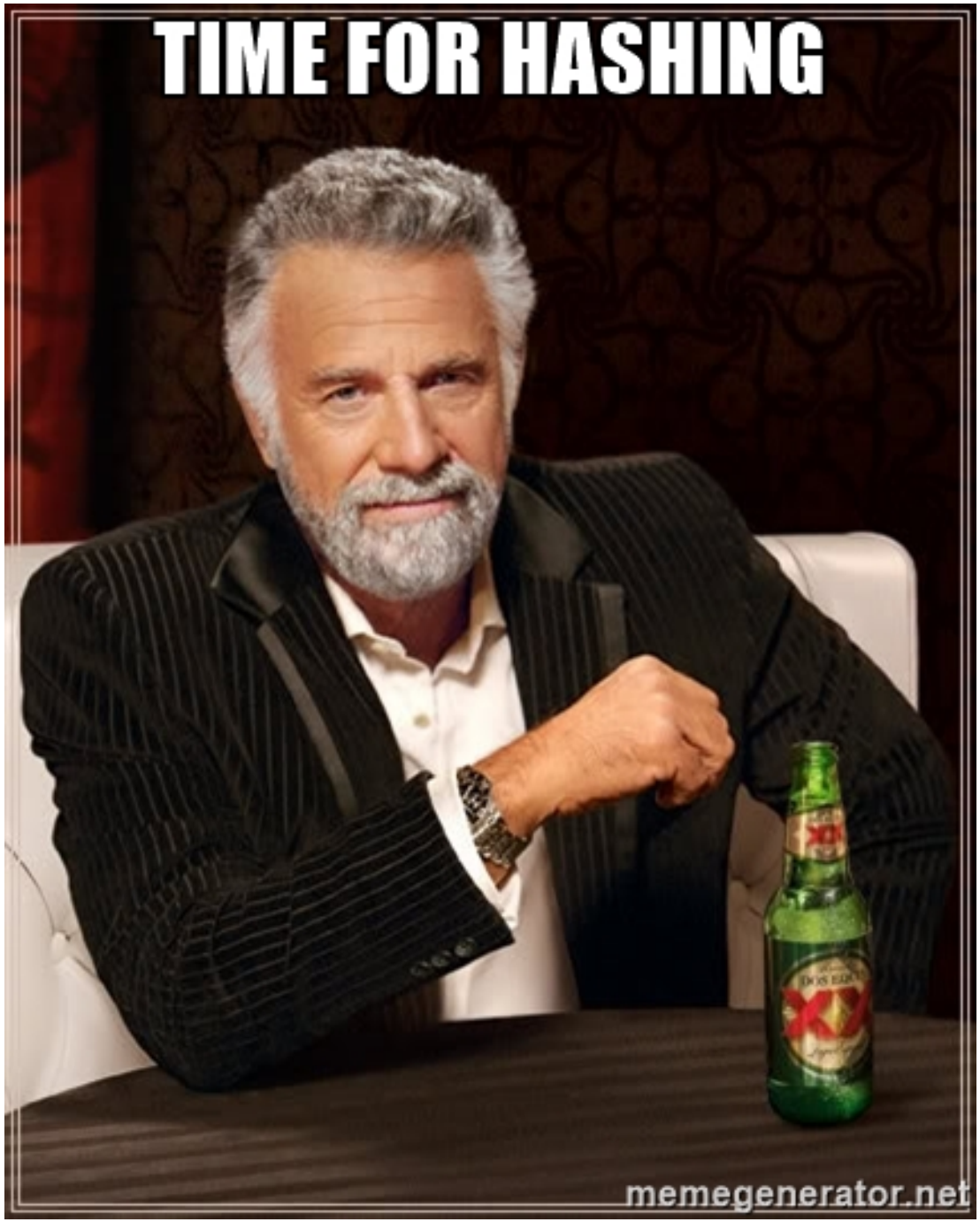
Two ways to prove this claim.

1. Chernoff and union bound
2. Binomials and union bound

$$\mathbf{Pr}\left[\exists i : X_i \geq \underbrace{\frac{3\log n}{\log\log n}}_{k}\right] \leq n\binom{n}{k}\frac{1}{n^k} \leq \frac{1}{n}.$$

Reminder, Chernoff bound: Let $X_1, \ldots, X_n$ be independent RVs with $X_i \in \{0,1\}$, $X = \sum_{i=1}^{n} X_i$, then:

$$\mathbf{Pr}\left[X \geq (1+\delta)\mathbb{E}\left[X\right]\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathbb{E}[X]}$$


TIME FOR HASHING
memegenerator.net

Dictionary problem

Universe $U = [u] = \{0, \ldots, u-1\}$

Set $S \subseteq U$, $|S| = n$, $|S| \ll U$

Goal: design a data structure that supports efficiently the following operations.

- $\textsc{Make}()$: Initializes an empty dictionary
- $\textsc{Insert}(x)$: Add element x in S
- $\textsc{Lookup}(x)$: Does x appear in S
- $\textsc{Delete}(x)$: Removes x from S, if present

Questions:

- Why not a linked list?
- Why not an array over U?

Python dictionary

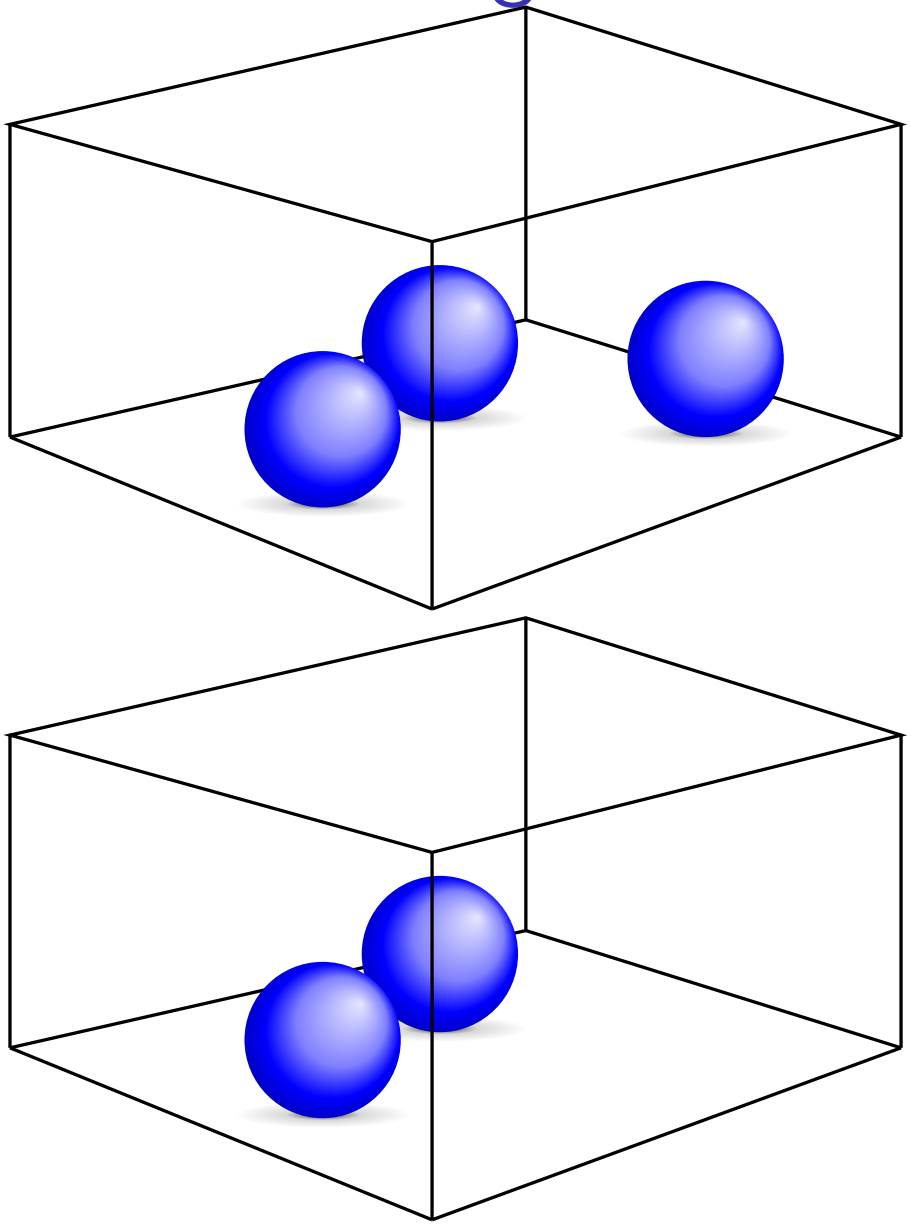
```
#empty table
d ={}

#insert
d["Andrei Rublev"] = "Tarkovsky"
d["Stalker"] = "Tarkovsky"
d["Viridiana"] ="Bunuel"
d[('123','a')] = "a123"

#lookup
print(d["Stalker"])
print(d[('123','a')])

#delete
del d["Stalker"]
print(d["Stalker"]) #KeyError: 'Stalker'
```

Hashing

- **Basic idea:** Work with an array of size $m = O(|S|)$ rather than of size $O(|U|)$!
- **Hash function:** $h : [u] \to [m]$
- **Hash table:** Array. We place $x \in S$ at position $h(x)$.
- **Collision:** $x \neq y \in U$ get mapped to $h(x) = h(y)$.

1. How do we choose h?
2. How do we resolve conflicts?

Balls and bins again : n balls, r bins



Problems: (1) Collision? (2) No empty bin? (Whiteboard)

Balls and Bins Revisited: k-wise independence

Consider the load of some bin.

$$\sum_{K \subseteq S, |S|=k} \frac{1}{r^k} \leq \left(\frac{en}{k}\right)^k r^{-k} = \left(\frac{en}{rk}\right)^k$$

- If $k > 2en/r > 2 \log r$ the probability of k balls in any single bucket is $< 1/r$.
- No need for full randomness, but randomness over all subsets of k hash values.

Source: See also Rasmus Pagh's slides

Balls and Bins Revisited: k-wise independence

Definition: RVs $X_1, \ldots, X_n$ are k-wise independent iff for any set of indices $i_1, \ldots, i_k$, RVs $X_{i_1}, \ldots, X_{i_k}$ are independent.

Definition: A set of hash function $\mathcal{H}$ is a k-wise independent family iff the random variables $h(0), \ldots, h(u1)$ are k-wise independent when $h \in \mathcal{H}$ is drawn uniformly at random.

Example 1: The set $\mathcal{H}$ of all functions from $[u]$ to $[m]$ is k-wise independent for all k.

 # Bits: $u \log m$ (u is enormous!)

2-wise independent family

Exercise: We can construct a 2-wise independent family as follows.

- p is prime
- a, b chosen uar from $[p]$
- The hash of x is

$$h(x) = ax + b \mod p,$$

How many bits do we need now?

Generalization: Polynomials with random coefficients


RANDOM
HASH FUNCTION
memegenerator.net

String hashing: bad choice, why?

```
unsigned long hash(unsigned char *str)
{
unsigned int hash = 0;
int c;

while (c = *str++)
    hash += c;
return hash;
}
```

String hashing: djbx33a

```
unsigned long hash(unsigned char *str)
{
    unsigned long hash = 5381;
    int c;
    while (c == *str++)
            hash = ((hash << 5) + hash) + c;
    return hash;
}
```


SO YOU ARE TELLING ME
THAT DJBX33A WORKS FINE?
memegenerator.net

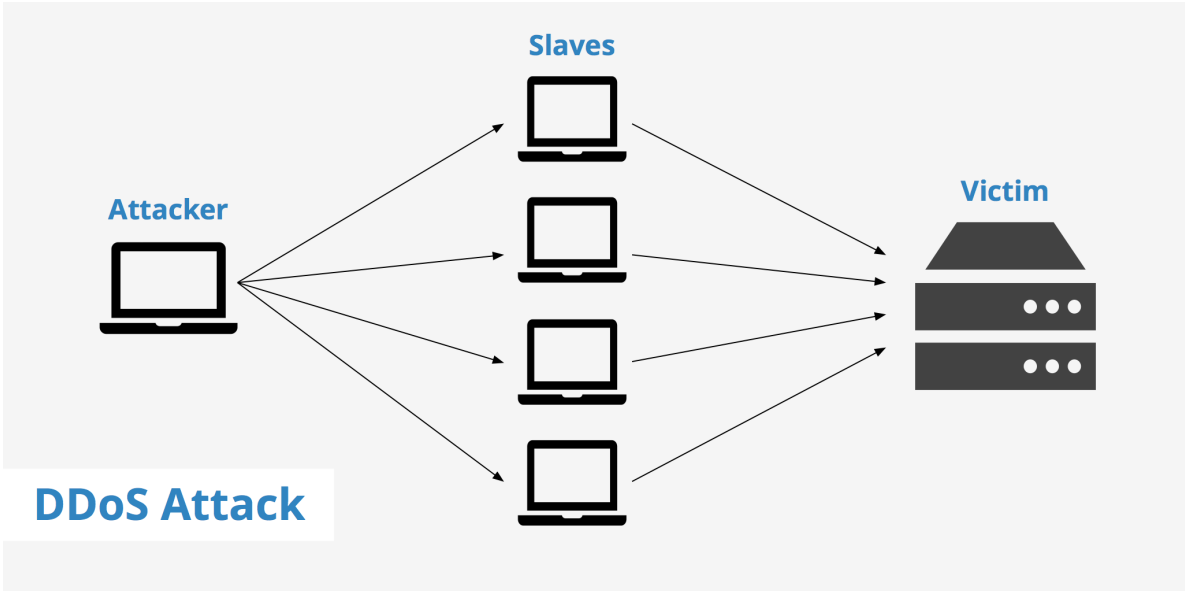
djbx33a is Vulnerable to attacks

```
#include <iostream>
#include <cstring>

// author: Charalampos Tsourakakis
// CS 591 T2, BU

unsigned long hash(std::string str){
    unsigned long hash = 5381;
    int c;

    for( int i = 0; i < str.length(); i++)
        hash = ((hash << 5) + hash) + str.at(i)
    return hash;
}
```

djbx33a is Vulnerable to attacks

```
int main()
{
    std::string s="Ey";
std::cout<<"h(Ey)="<<hash(s)<<std::endl;
s = "FZ";
std::cout<<"h(FZ)="<<hash(s)<<std::endl;
return 0;
}

>> g++ -o DoSdjbx33a DoSdjbx33a.cc
>> ./DoSdjbx33a
h(Ey)=5862307
h(FZ)=5862309
```

Hash-flooding DoS



Definition: Send to a server many inputs with a same hash (enforces linear)

Hash-flooding DoS

- Verify (in the scribe too!) that $h(Ey) = h(FZ)$ for djbx33a hash function.
- In one of the project problems you will create hash attacks for Java's hashCode() function.
- We discussed in the first lecture fingerprinting. In general, this is a way to turn different types of inputs into integers.
- Then, frequently these fingerprints are used as keys to hash tables.

String hashing: java.lang.String.hashCode()

```
unsigned long hash(unsigned char *str)
{
    unsigned long hash = 0;
    int c;
    while (c == *str++)
            hash = ((hash << 5) - hash) + c;
    return hash;
}
```

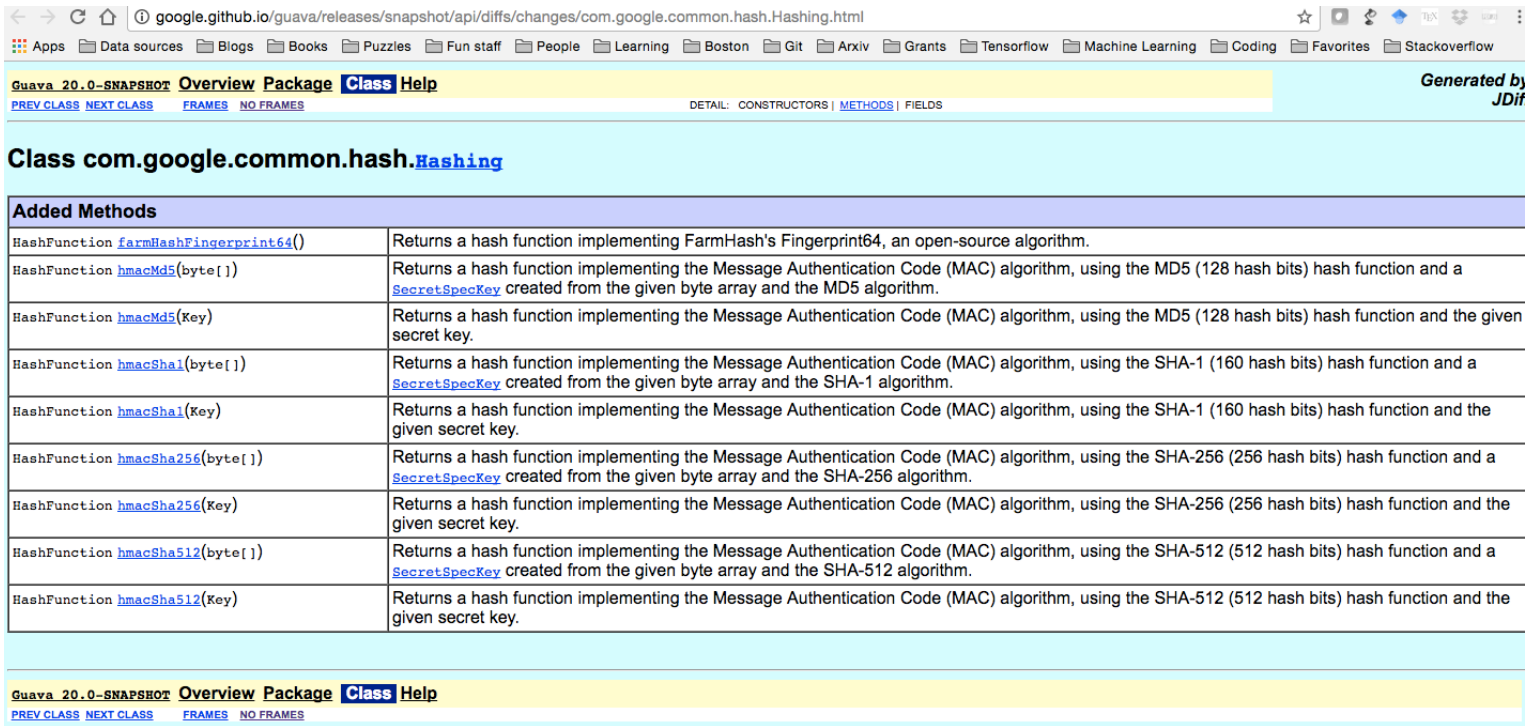
Hash-flooding DoS

Here is what your website may look like after a successful Denial of Service Attack:

Figure from: How to Detect a Denial of Service (DoS) Attack

Hash-flooding DoS

For example: FARMHASH::FINGERPRINT64() takes as input a *string*, and outputs a *uint64*. [Not secure!]



google.github.io/guava/releases/snapshot/api/diffs/changes/com.google.common.hash.Hashing.html

Apps | Data sources | Blogs | Books | Puzzles | Fun staff | People | Learning | Boston | Git | Arxiv | Grants | Tensorflow | Machine Learning | Coding | Favorites | Stackoverflow

Guava 20.0-SNAPSHOT Overview Package Class Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES DETAIL: CONSTRUCTORS | METHODS | FIELDS

Generated by JDif

Class com.google.common.hash.Hashing

Added Methods	
HashFunction farmHashFingerprint64()	Returns a hash function implementing FarmHash's Fingerprint64, an open-source algorithm.
HashFunction hmacMd5(byte[])	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the MD5 (128 hash bits) hash function and a SecretSpecKey created from the given byte array and the MD5 algorithm.
HashFunction hmacMd5(Key)	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the MD5 (128 hash bits) hash function and the given secret key.
HashFunction hmacSha1(byte[])	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the SHA-1 (160 hash bits) hash function and a SecretSpecKey created from the given byte array and the SHA-1 algorithm.
HashFunction hmacSha1(Key)	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the SHA-1 (160 hash bits) hash function and the given secret key.
HashFunction hmacSha256(byte[])	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the SHA-256 (256 hash bits) hash function and a SecretSpecKey created from the given byte array and the SHA-256 algorithm.
HashFunction hmacSha256(Key)	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the SHA-256 (256 hash bits) hash function and the given secret key.
HashFunction hmacSha512(byte[])	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the SHA-512 (512 hash bits) hash function and a SecretSpecKey created from the given byte array and the SHA-512 algorithm.
HashFunction hmacSha512(Key)	Returns a hash function implementing the Message Authentication Code (MAC) algorithm, using the SHA-512 (512 hash bits) hash function and the given secret key.

Guava 20.0-SNAPSHOT Overview Package Class Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

For secure cryptographic functions, a good start is the MD5 algorithm.