Recap

<u>Thm</u>: The max-loaded bin $O\left(\frac{\log N}{\log \log N}\right)$ balls with prob. at least $1 - \frac{1}{N}$.

<u>proof</u>: <u>High level steps</u>:

1. prob. of any bin receiving at least $O\left(\frac{\log N}{\log \log N}\right)$ balls
   → want: $1/N^2$ (for union bounding over bins)
2. prob. of there being <u>at least one</u> bin with at least these many balls.
   → want: $\frac{1}{N}$

<u>union bound</u>:

$p(A \cup B \cup C \cdots) \leq p(A) + p(B) + \cdots$

Recap

$$p(\text{bin } i \text{ has at least } k \text{ balls}) \leq \binom{N}{k}\left(\frac{1}{N}\right)^k$$

$$= \frac{N!}{(N-k)!\,k!}\,\frac{1}{N^k}$$

$$\leq \frac{N^k}{k!}\cdot\frac{1}{N^k}$$

$$= \frac{1}{k!}$$

Sterlings approx. $\quad k! \sim \sqrt{2\pi k}\left(\frac{k}{e}\right)^k$

Choose $\quad k = O\left(\frac{\log N}{\log\log N}\right)$ gives desired result

Clarification:

Sets $S_1 \quad S_2 \quad \cdots \quad S_{\binom{N}{k}}$

Events $E_1 \quad \cdots \quad E_{\binom{N}{k}}$

$E_j :=$ Set S_j landing in bin i

E = bin i getting at least k balls

$= E_1 \cup E_2 \cdots \cup E_{\binom{N}{k}}$

$$P(E) = P\left(E_1 \cup \cdots \cup E_{\binom{N}{k}}\right)$$

$$\leq \sum P(E_i)$$

power-of-2-choice:

$$O(\log \log N)$$

proof (intuition/sketch):

height (b) = num. of balls in its bin after placing b

Prob. of an incoming ball getting height 3 is at most?

Fraction of bins that can have ≥ 2 balls?

- at most $1/2$

N balls
N bins

$$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

Expected # of bins with 3 balls is at most $= \frac{N}{4}$

Same calc. for height 4 :

$$\frac{1}{4} \cdot \frac{1}{4} = \frac{1}{2^{2^{4-2}}}$$

prob. of an incoming ball getting height h is at most

$$\frac{1}{2^{2^{h-2}}}$$

Choose $h = O(\log \log N) + 2 \Rightarrow$ prob. $:= \frac{1}{N}$

power of d-choices

<u>Thm</u>: for any $d \geq 2$, for d-choice process, max load

$$\frac{\log \log N}{\log d} \pm O(1)$$

with p. atleast $1 - O(1/N)$

<u>Diminishing returns</u>.

Data streaming model :

elements coming as a "stream"
Limited storage (cannot store all elements)

Notation :
Elements in the stream $a_1, a_2 \ldots$ from alphabet U

Ex. of functions:

1. Sum (easy)
2. Max (easy)
3. Median (tricky)
4. Heavy-hitters i.e 'most appeared'
5. Num. of distinct elements

Natural option: Sampling

E.g.: # unique elements

Sample 10% → count uniques → mult by 10.

Can lead to incorrect answer

Counter. ex:

Stream of len. n

$n/2$ of uniques. $\frac{n}{4}$ appear twice

10% Sampling:-

$$\# \text{ uniques} = 0.1 * \frac{n}{2} + \frac{n}{4}\left(2 * 0.1 - 0.1^2\right)$$

$$\simeq \frac{n}{10}$$

So finally # uniques $= \frac{n}{10} \cdot 10 = n$

Abstraction :

Stream at time t as a vector $x^t \in \mathbb{Z}^{|U|}$

$$x^t = (x_1^t \; x_2^t \; \dots \; x_{|U|}^t)$$

number of time element 1 of U has been seen until time t

Generalization :- (add e) (del e)

Ex: $U = \{A, B, C\}$

add A add B del A

$(0,0,0)$ $(1,0,0)$ $(1,1,0)$ $(0,1,0)$ . .

Assumption :- # del $\leq$ # add ?

Heavy hitters:

ϵ-heavy hitters: indices i s.t $x_i > \epsilon \|x\|_1$

Count-query:

At time t, for any index i

output an estimate

$$y_i \in x_i \pm \varepsilon \|x\|_1$$

For ε-heavy hitter: can look at i s.t $y_i > 0$

Hashing-based solⁿ: Count-min Sketch

Step 1: $h : U \to [M]$

Array $A[1 \ldots M]$ can store non-negative integers

If add i

$A[h(i)]++$

Else (del i)

$A[h(i)]--$

Estimate for x_i^t : $y_i = A[h(i)]$

count query

$$A[h(i)] = \sum_{j \in U} x_j^t \, \underbrace{\mathbb{1}\{h(i) = h(j)\}}_{\text{indication}}$$

$$= \underbrace{x_i^t}_{\text{interested in}} + \sum_{j \neq i} x_j^t \, \mathbb{1}\{h(i) = h(j)\}$$

Error in estimate =

Assume universal hash family.

h from

Universal

$$p(h(x_i) = h(x_j)) \leq \frac{1}{M}$$

$$E\left[\sum_{j \neq i} x_j^t \, \mathbb{1}\{h(j) = h(i)\}\right]$$

$$= \sum_{j \neq i} x_j^t \, P\big(h(j) = h(i)\big)$$

$$\leq \sum_{j \neq i} x_j^t \, \frac{1}{M}$$

($\because$ universal hash family)

$$= \frac{1}{M}\left(\|x^t\|_1 - x_i^t\right)$$

$$\leq \frac{\|x^t\|_1}{M}$$

$\varepsilon \|x^t\|_1$

$$\varepsilon = \frac{1}{M}$$

Step 2:

Boost success probability

Idea/approach: Repeat & take the best outcome

ℓ hash functions $h_1 \dots h_\ell$

ℓ arrays $A_1 \dots A_\ell$

Same approach as earlier.

$$y_i = \min_k A_k[h_k(i)]$$

($\because$ overestimate)

$y_i = A[h(i)]$

now

$A_1[h_1(i)], A_2[h_2(i)]$

$\dots A_\ell[h_\ell(i)]$