大数据计算基础

第三章 面向大数据的数据结构

哈尔滨工业大学
刘显敏
liuxianmin@hit.edu.cn

预览

- 谷歌/淘宝是怎么做下面这些事情的
  - 取样
    - 比例取样
    - 固定size取样
  - 频度统计
    - 统计item发生的次数
  - 白名单过滤
  - 统计不同查询的个数
  - 评估用户访问的均匀性
  - 发现最热item
- 简单的数据统计问题，在大数据场合下，新的方法

流

- 数据以流的方式进入
  - 搜索引擎的查询请求
  - 微博更新
- 特点
  - 无穷
  - 非平稳
    - 流的到达速率取决于用户行为，系统无法控制
- 元素（Element）
  - Tuple

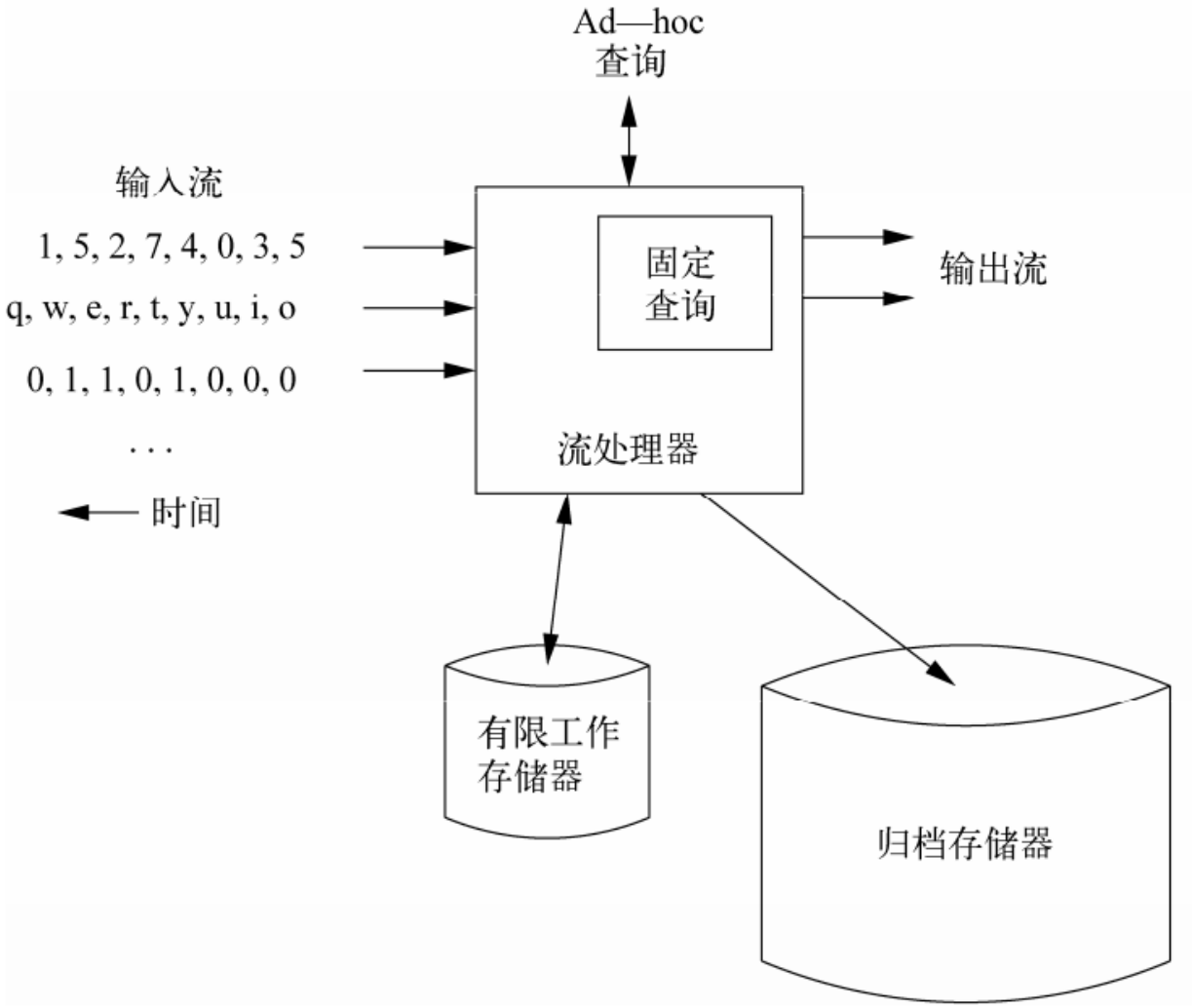
大数据下的系统限制

- 流源源不断地来
  - 要求实时处理
- 系统限制
  - 存储限制，不能存这么多
  - 存得多，处理量也大，处理能力限制
- NSA（美国棱镜门）
  - 存几个月
- 流处理
  - 有限存储情况下，怎么实时处理？
  - Online learning

模型

两种查询

1. 固定查询：
   - Standing query
   - 从不停止
   - 例：
     - 历史最高温度
   - 事先写好
2. Ad-hoc查询
   - 不全存，但还存一些内容
   - 根据这些存储内容应答



问题

- 取样：
  - 随机取样（Sampling）
  - 过滤（白名单）：选取特定属性的元素（Filtering）
- 计数（一定窗口内）
  - 有多少个不同的元素？（distinct elements)
  - 各元素的Popularity?
    - 特征：各阶矩
  - 谁最流行?

应用

- Google：
  - 查询流
  - 发现最流行的查询关键字
- Yahoo：
  - 发现最流行的页面
- 微博：
  - 发现最热的话题
  - 找人
- 传感器网络
- 电话记录
  - 美国，棱镜门
- 网络交换机
  - 流量统计，优化路由
  - 检测DDoS攻击

抽样

- 两种抽样
  - 固定比率抽样
    - 1 in 10
  - 固定Size抽样
    - 总是保持s个元素

固定比率抽样

- 应用场合
  - 搜索引擎，一个用户的搜索中，重复的有多少？
  - 存不了全部，可以存1/10
- 最明显的办法
  - 每来一个query
  - 生成一个随机整数：0…9
  - 如果是0，就存起来
  - 1/10的采样
  - 然后统计其中的用户重复搜索比例
- 对吗？

有问题

- 假设：一个用户所有搜索字符串中，x个查询了一次，d个查询了两次，没有其他查询。
- 重复查询占比：d/(x+d)
- 随机采样10%后，重复查询占比是怎样的？
  - 采样后，获得(x+2d)/10个查询，其中x/10个查询是属于x，肯定只出现一次
  - 针对d的2d/10个查询
    - d中任一查询，两次都被抽中的概率为1/10×1/10 = 1/100
    - 所以，平均有d/100个查询会被抽中两次，占2d/100个查询
    - 剩下2d/10 – 2d/100 = 18d/100次查询，也只出现一次。
- 结果 $\frac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}} = \frac{d}{10x+19d}$
- 不等于d/(x+d)。错误

正确方法：按用户采样

- 挑1/10的用户，观察它们的全部查询
- 采样方法
  - Hash(User ID) mod 10，把用户分到十个桶中
  - 选第一个桶的用户（hash后结果为0）
- 挑2/10的用户怎么办？
  - 选前面两个桶

固定Size抽样

- 总是保持s个元素
  - 这s个元素，是对过去所有元素的均匀取样
  - 即：过去所有元素，进入这s个元素的概率相同
- 直观方案：
  - 全存起来，然后从中随机挑s个
- 大数据下，因为存储空间的限制，不可行
- 流方案
  - 进来一个新元素时，
    - 新元素以概率p进入s
    - 原有的s个元素按一定的概率从s中剔除

新元素进入S的概率p

- 假设已到达n个元素，它们以s/n的概率被采样，组成s个元素的集合
- 新进来一个元素，一共到达了n+1个元素。
  - 这n+1元素，以相同概率进入s
    - 这个概率： s/(n+1)
  - 所以，这个新元素以s/(n+1)的概率进入s

• p = s/(n+1)

S中原元素的剔除策略

- 原来在s个元素集合中的元素，随机剔除一个
- 不被剔除的概率

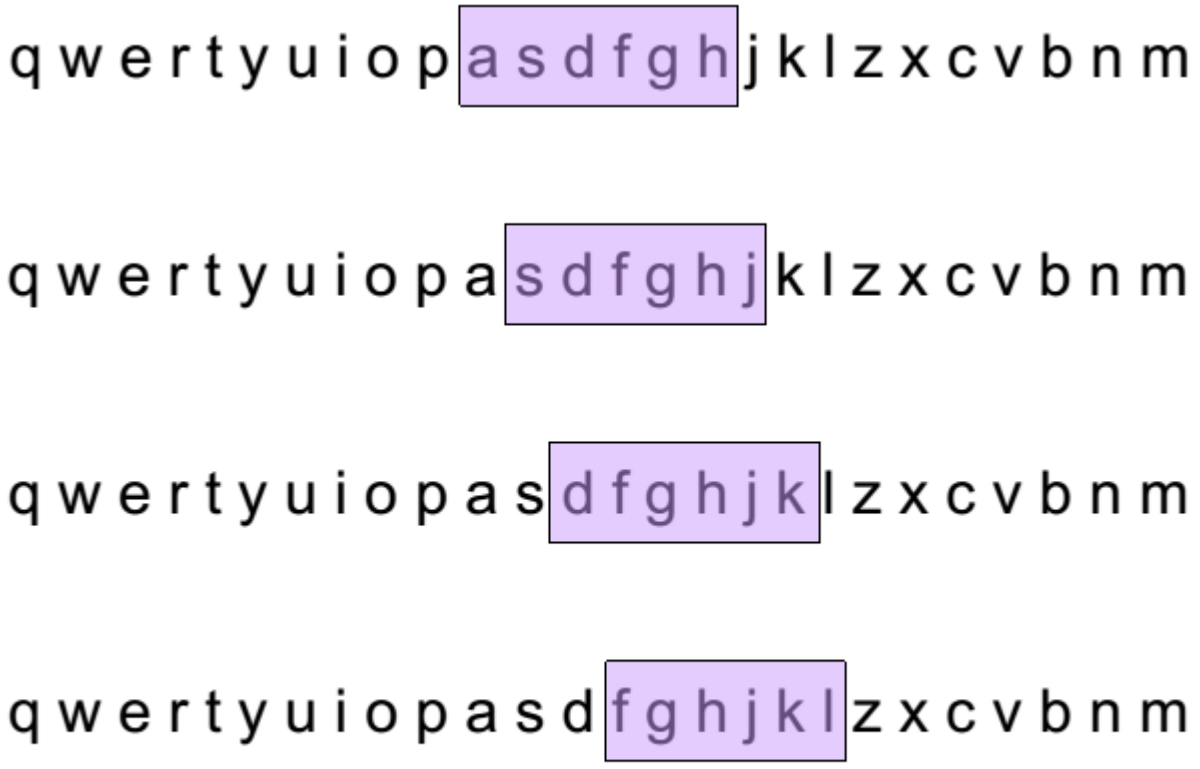
$$(1-\frac{s}{n+1})+(\frac{s}{n+1})(\frac{s-1}{s})=\frac{n}{n+1}$$

新元素不进s的概率　　新元素进s，但在s中不被剔除的概率

- 原先，这n个元素，是以s/n概率进入s的。
- 这一轮过后，任一元素留在s中的概率 $\frac{s}{n}\cdot\frac{n}{n+1}=\frac{s}{n+1}$
- 和新到元素的留下概率s/(n+1)相等
- 结果：所有n+1个元素，以s/(n+1)的概率留下

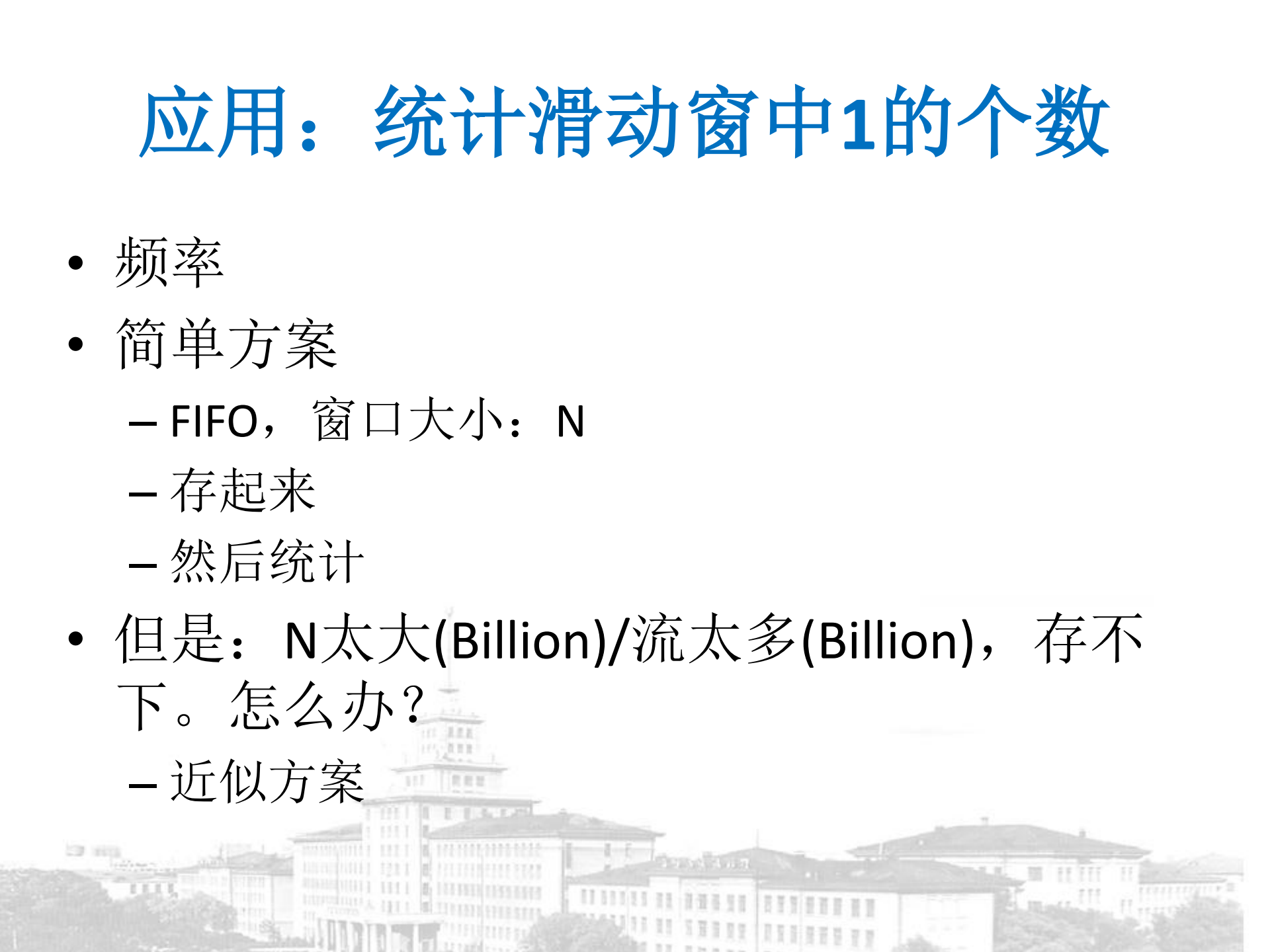
示例

- N = 6


q w e r t y u i o p a s d f g h j k l z x c v b n m
q w e r t y u i o p a s d f g h j k l z x c v b n m
q w e r t y u i o p a s d f g h j k l z x c v b n m
q w e r t y u i o p a s d f g h j k l z x c v b n m
Past
Future

应用：统计滑动窗中1的个数

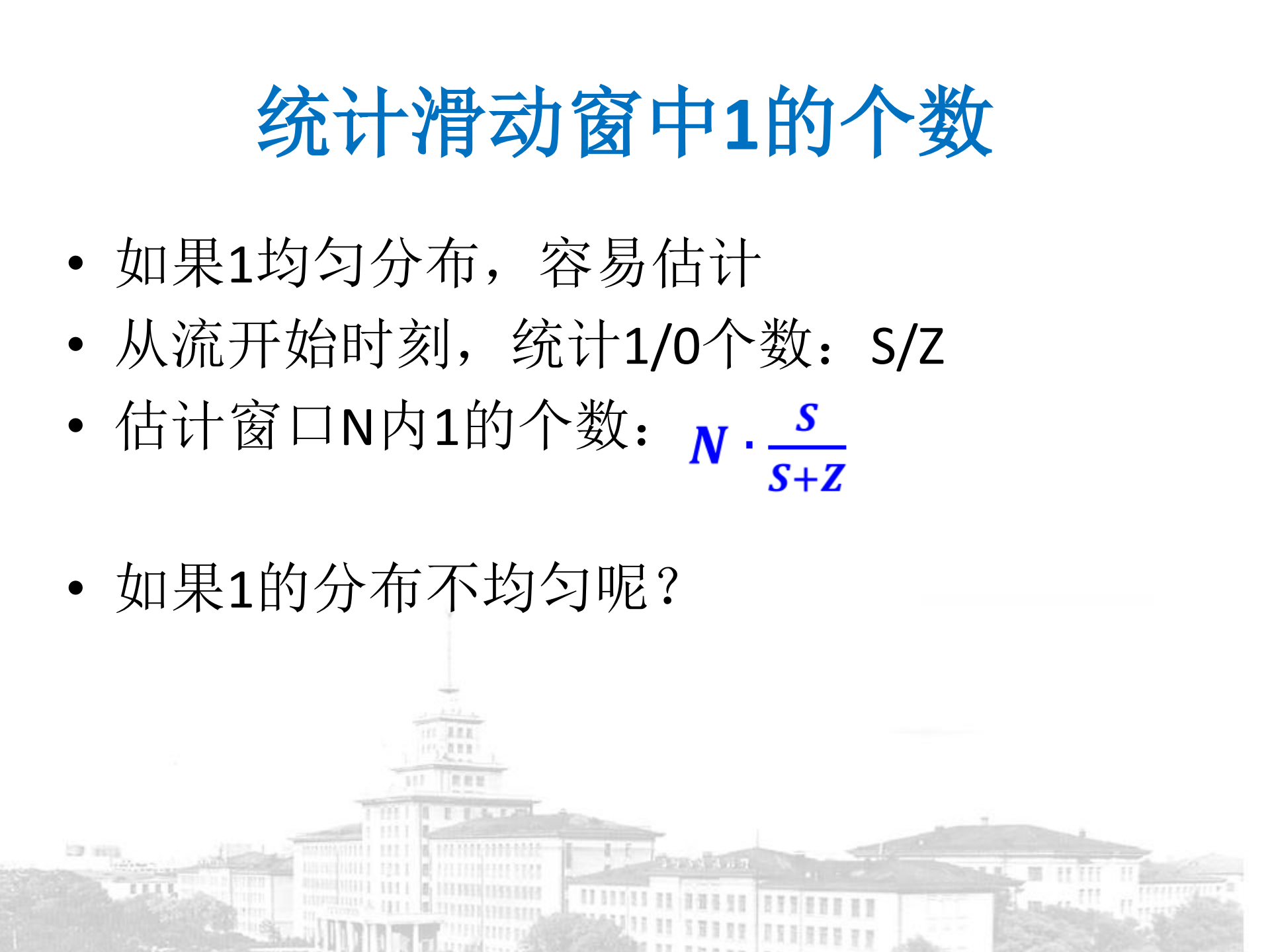
- 频率
- 简单方案
  - FIFO，窗口大小：N
  - 存起来
  - 然后统计
- 但是：N太大(Billion)/流太多(Billion)，存不下。怎么办？
  - 近似方案



统计滑动窗中1的个数

- 如果1均匀分布，容易估计
- 从流开始时刻，统计1/0个数：S/Z
- 估计窗口N内1的个数：$N \cdot \frac{S}{S+Z}$

- 如果1的分布不均匀呢？

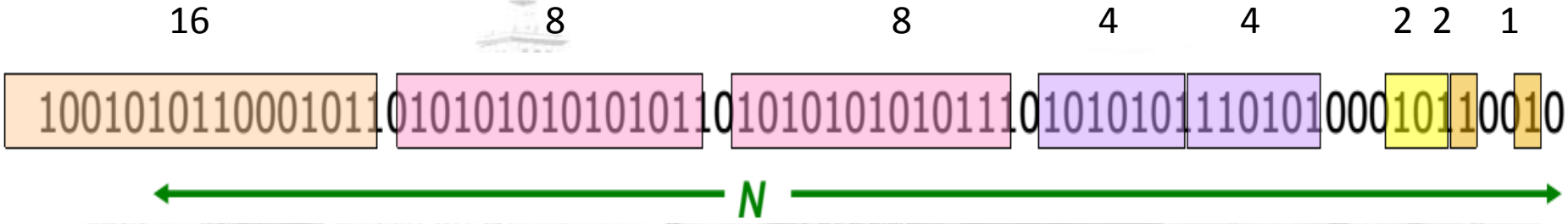


DGIM方法

- 每个流，存储 $O(\log^2 N)$ 比特
- 结果误差不超过正确结果的50%
  - 可以进一步减少

DGIM

- [Datar , Gionis, Indyk, Motwani]
- 指数窗口
- 每个窗口中包括 i 个1， i : 2的幂（指数增长）
- 同样i的窗口最多可以有两个
- 窗口不重叠，可以不连续（中间可以隔0）



DGIM需要的存储空间

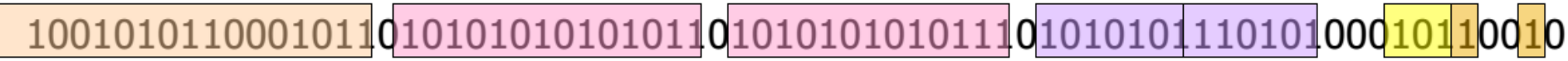
- 每个子窗（Bucket）有一个时标，记录结束时间
  - 取值范围 1 … N
  - 需要 $O(log_2 N)$ 比特存储空间
- 每个bucket记录自己包含的1的个数
  - 取值范围：1…logN
  - 需要 [O(log log *N*) bits] 存储空间

更新

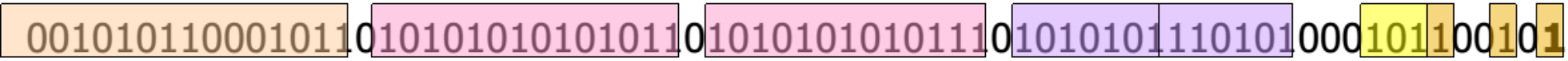
- 新元素到了
- 如果一个Bucket的end time已超过当前时刻 - N，drop它
- 如果新元素是0，什么也不做
- 如果是1
  - 创建一个Bucket，size = 1， end time = 当前时间
  - 如果有3个1，就合并前两个2。
  - 依次类推，如果有3个一样的小的，就合并前两个为一个大的。
    - 雪崩式前滚

示例

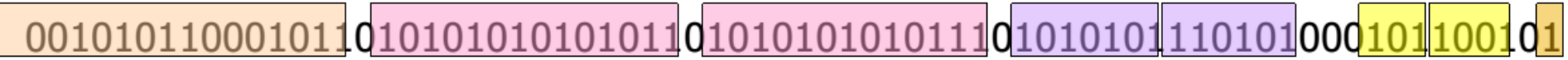
Current state of the stream:



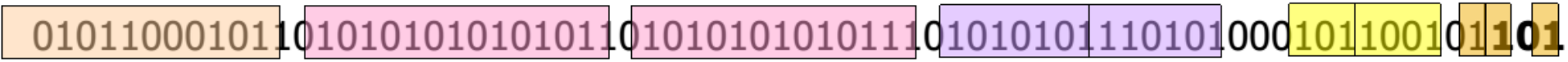
Bit of value 1 arrives



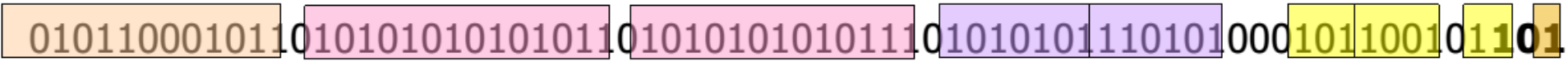
Two orange buckets get merged into a yellow bucket



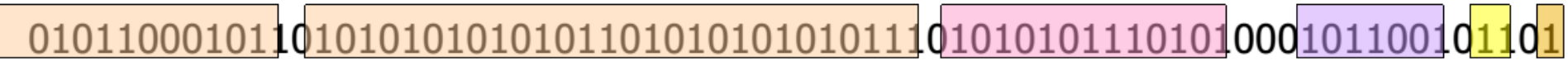
Bit 1 arrives, new orange bucket is created, then 0 comes, then 1:



Buckets get merged…

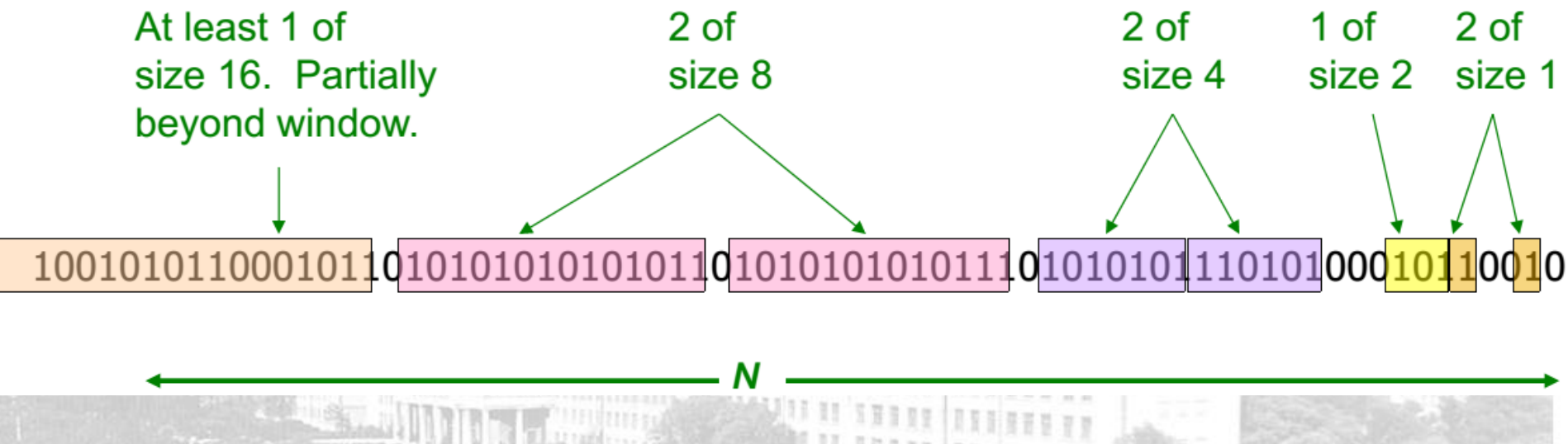


State of the buckets after merging



估计1的个数

- 除了最后一个bucket，把其他bucket的size相加
  - Size就是其中1的个数
- 再加上最后一个Bucket size的一半
  - 因为最后一个bucket，只是最后一位还在N里，



Error bound：50%

- 假设最后一个bucket的size：2^r
- 我们在统计中算了它的一半“1”，所以，最多产生2^{r-1}的错误
- 比它size小的bucket有2^{r-1}，2^{r-2}，2^{r-3}，…，1，每种至少有一个
- 所以，它们包含的“1”的个数至少为：$2^{r-1} + 2^{r-2} + 2^{r-3} + \ldots + 1 = 2^r - 1$.
- 最后一个bucket在窗口中至少还有1个“1”，所以，“1”的个数至少为2^r
- 所以，最大的错误率：$2^{r-1} / 2^r = 1/2 = 50\%$

扩展

- 同样size的bucket数目可以是r或r-1个。 r > 2
- 最大Size的bucket，可以有1,…,r个
- 错误的上界1/(r-1)
- 实践中，根据需要选择r

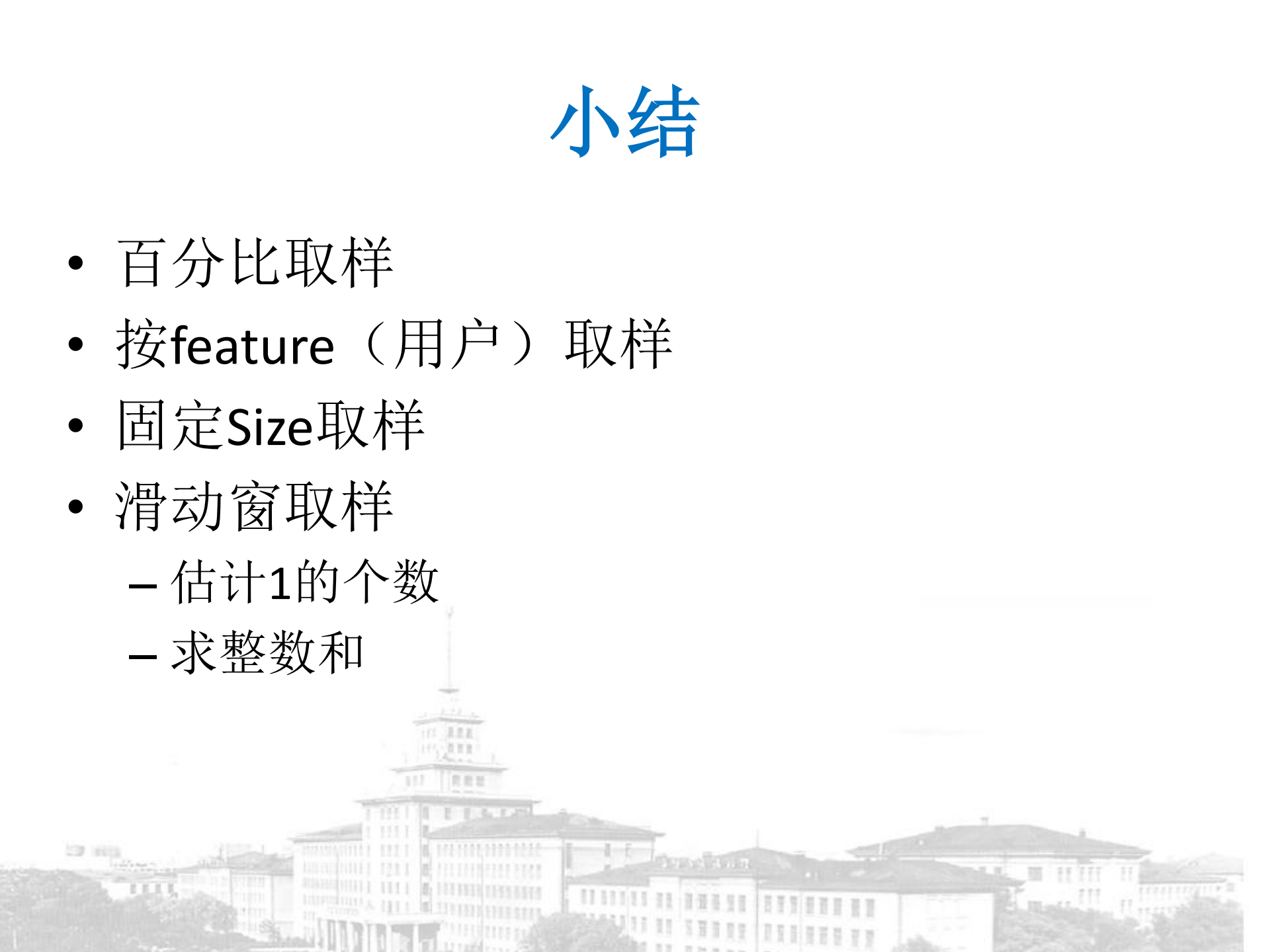
应用：窗口内整数的和

- 把整数的每一个bit作为一个stream
- 统计每一个stream的1的个数，C_i
- 求和：

$$\sum_{i=0}^{m-1} c_i 2^i$$

小结

- 百分比取样
- 按feature（用户）取样
- 固定Size取样
- 滑动窗取样
  - 估计1的个数
  - 求整数和



Bloom filter

- Bloom是一个人
- 从stream中选择符合特定条件的元素
- 例1：垃圾邮件检查
  - 白名单
- 例2：Google Alert
  - Pub-Sub系统，每个人可以设定订阅的关键词
- 明显的方法
  - 建立Hash表，查询，命中
- 大数据下，filter太多，数据太多，怎么办？
  - 包括10 billion 个白名单

初始化

- 白名单中包括s个允许的key值
  - s = 1 billion
- n个检查位，n >> s，初始化为0
- 把这s个白名字Hash到1,…,n上
  - 对应的bit位设1
- 最后，n中大约有s个“1”
  - 事实上小于s个，因为会重合。

到底有几个1？

- 一个白名字，被均匀地撒在n个比特上
  - 撒上概率：1/n
- 一个比特位，没有被撒上的概率
  - 被1个白名字错过的概率：$1 - 1/n$
  - 被所有s个白名字都错过的概率
    - $(1-1/n)^s = (1-1/n)^{n(s/n)}$
      - 近似等于 $e^{-s/n}$
- 所以，一个比特位，被撒上的概率
  - $1 - e^{-s/n}$
- 总共，$n(1 - e^{-s/n})$个比特位被撒上
  - 值为“1”

检查

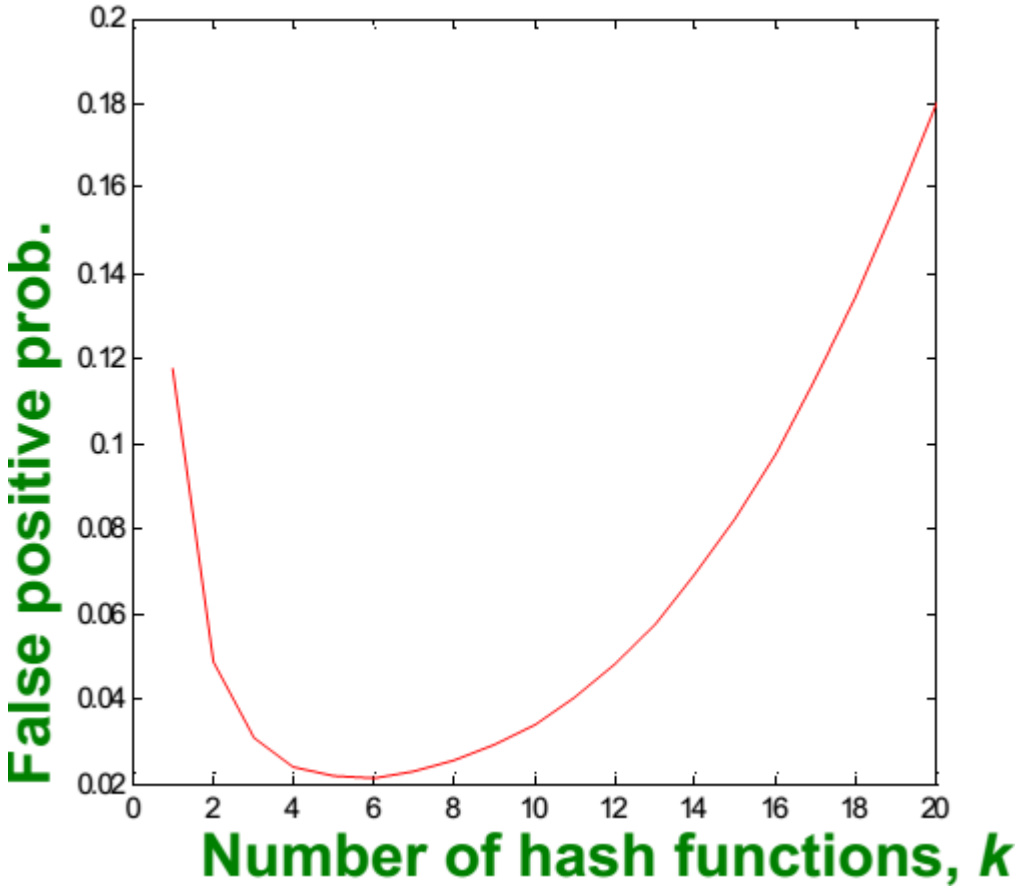
- 来了一个邮件，把发件人地址，hash一下，如果对应的比特位为0，肯定不在白名单里，Reject
- 不在白名单里，也会被均匀撒在n个比特位上
  - 如果那个比特位碰巧是“1”，就会pass
  - False positives - 假阳（FP）
    - Pass：Positive
- 和n中“1”的比例有关，
  - $n(1 - e^{-s/n})/n = 1 - e^{-s/n}$
- 所以，可以通过增加n，降低FP概率
  - $s = 10^9$, $n = 8 \times 10^9$，概率 $1 - e^{-1/8} = 0.1185 \sim 1/8 = s/n$

改进：多个hash函数

- 初始化
  - 对s中任一元素，用k个独立hash函数，分别撒k次
  - “1”的个数：
    - 类似前面，只是撒了ks次
    - $n(1 - e^{-ks/n})$
- 检查
  - 来一封信，用这k个hash检查，全部为“1”才行。
  - False positive率
    - 混过去一个hash函数，概率$(1 - e^{-ks/n})$
    - 混过去全部k个hash检查，概率$(1 - e^{-ks/n})^k$
- K=2， 概率 0.0493 ~ 1/20 << 1/8
  - 改进了性能

K的选择

- K不是越大越好
- 对这个例子，最优的在6的样子。

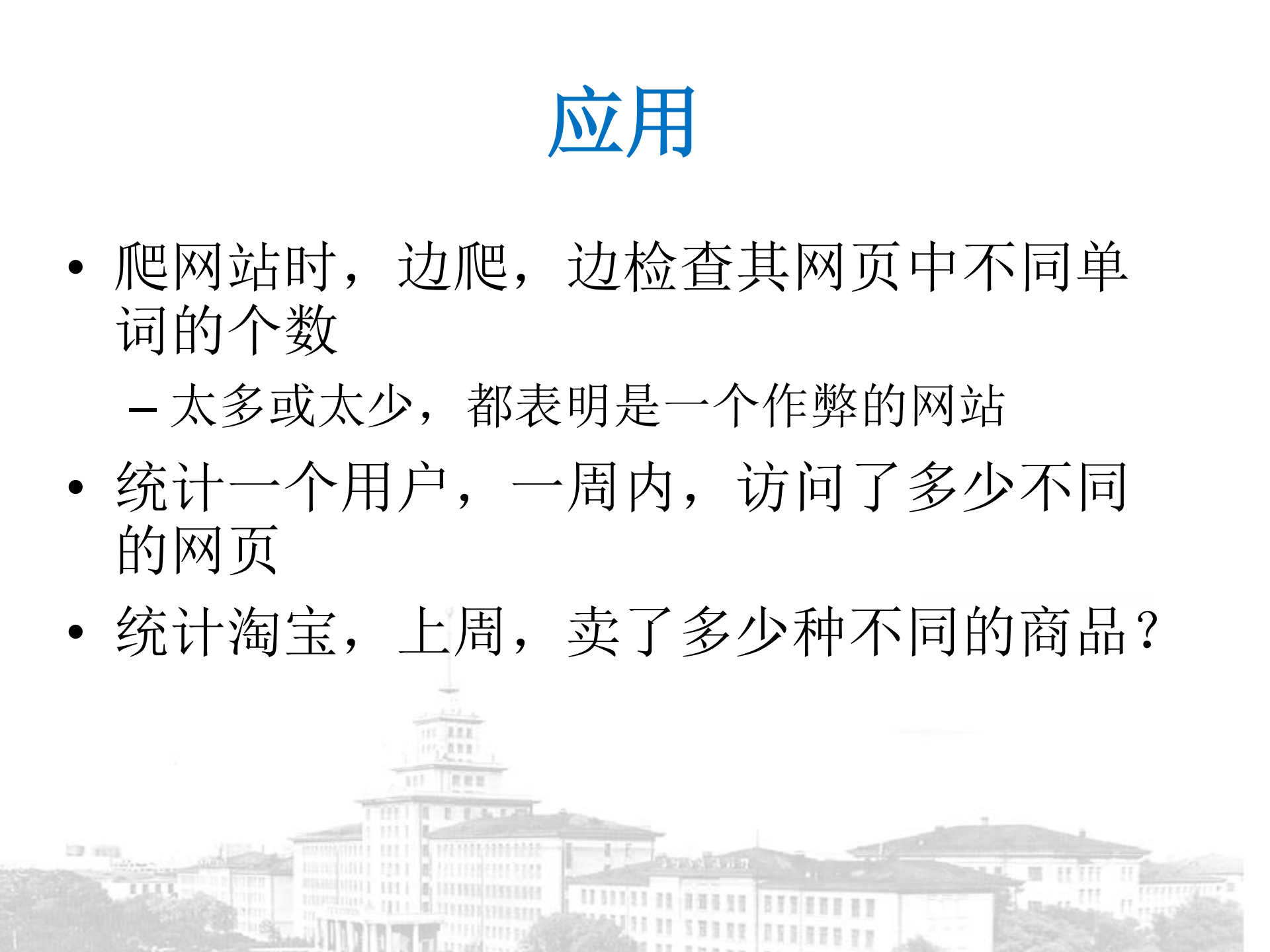


Bloom Filter总结

- 只会false positive
- 不会false negative
  - 错杀概率 = 0
- 适合预处理
  - 先筛选一些
- 适合硬件实现
- 适合并行
  - Map-reduce

应用

- 爬网站时，边爬，边检查其网页中不同单词的个数
  - 太多或太少，都表明是一个作弊的网站
- 统计一个用户，一周内，访问了多少不同的网页
- 统计淘宝，上周，卖了多少种不同的商品？



明显的方法

- 建立一个Distinct元素列表（hash表）
- 进来一个，和列表中已有的元素对照，如果不同，就加入
- 跟踪列表Size的变化

大数据情况下

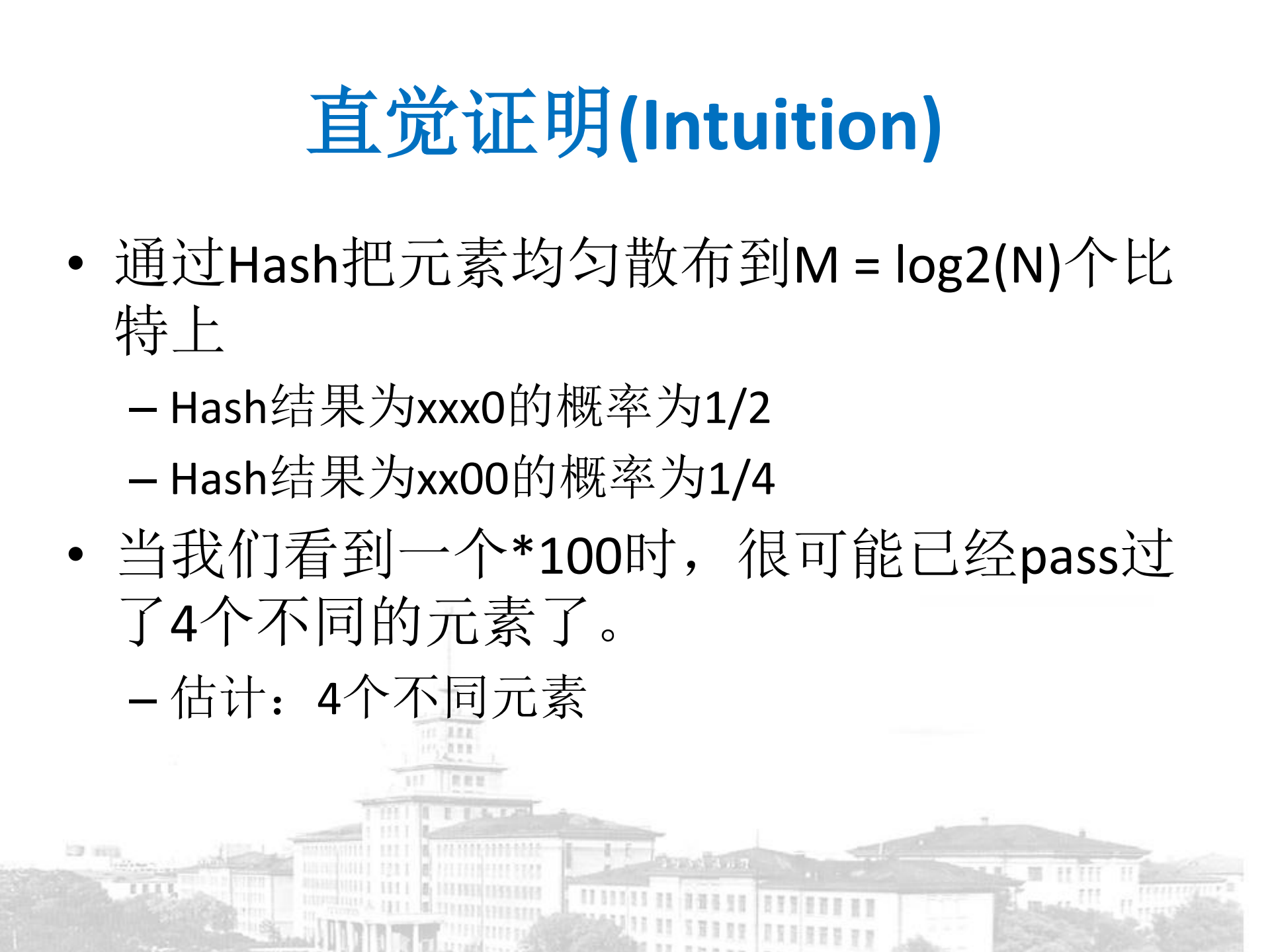
- 存不下
- 维护成本很高
- 需要
  - 减少存储要求
  - 减小计算复杂度
- Tradeoff：
  - 准确性 <> 实用性
- 估计

Flajolet-Martin方法

- 启发式算法
- 用Hash，把N个元素，映射到至少log2(N)比特上
- 检查映射的结果，看它们尾部连0的个数：r_i
  - 例：1100 -> 2
  - 1000 -> 3
- 找出最大的 r_i
  - 例：$R = \max\{2,3\} = 3$
- 估计不同元素个数为2^R
  - 例：$2^3 = 8$

直觉证明(Intuition)

- 通过Hash把元素均匀散布到M = log2(N)个比特上
  - Hash结果为xxx0的概率为1/2
  - Hash结果为xx00的概率为1/4
- 当我们看到一个*100时，很可能已经pass过了4个不同的元素了。
  - 估计：4个不同元素



更形式化的证明

- 一个元素，hash后，尾部连续r个0的概率
  - $(½)^r = 2^{-r}$
- m个不同元素hash后的m个结果，尾部都不“连续r个0”
  - 概率：$(1 - 2^{-r})^m = \left((1 - 2^{-r})^{2^r}\right)^{m2^{-r}}$

$$= e^{-m2^{-r}}$$

- 出现连续r个0的概率 $1 - e^{-m2^{-r}}$
  - $m >> 2^r$，概率为1，即总能得到连续r个0的结果。
  - $m << 2^r$，概率为0，即得不到连续r个0
- 所以，估计$m = 2^r$大致上是合理的。

实际应用

- 问题：
  - R加1，2^R就涨一倍。
  - E[2^R]无穷大
- 解决办法
  - 用多个hash函数，结果组合起来
- 组合办法
  - 平均：偶尔一个大值对结果的影响很大，不好
  - 中值：估计的结果总是2的幂次，取值不连续，也不好
- 解决方案：
  - 样本分组
  - 组内取平均
  - 组间取中值

矩估计

- *N*个到达的元素，统计各不同元素的流行度(Popularity)
  - 不同元素的集合A，
  - 各不同元素*i*出现的次数m_i（流行度）
- 流行度的K阶矩

$$\sum_{i \in A} (m_i)^k$$

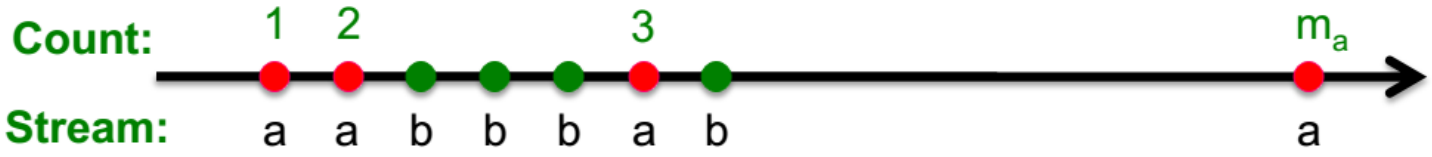
- 物理意义
  - k = 0，A的size，即不同元素的个数。 |A|
  - k = 1，N，stream长度，元素个数
  - k = 2，Surprise number（奇异数），Popularity分布均匀的度量

Surprise number（奇异数）

- Popularity分布的均匀程度的度量
- 例：
  - |A| = 11：11个视频
  - N = 100：100次用户观看
- 观看在视频上的分布
  - Case 1：10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9
    - Surprise S = 910
    - 比较均匀
  - Case 2：90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1
    - Surprise S = 8110
    - 更不均匀

AMS方法

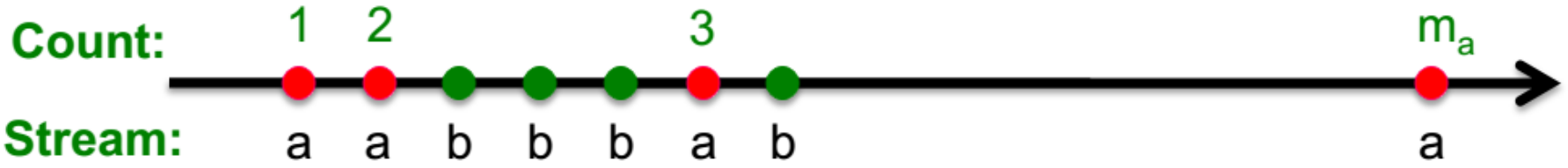
- Alon, Matias, and Szegedy
- 以k = 2为例
- 随机挑一个时刻，对stream采样
  - 采样获得的值存在X.el里
- 然后对后面进来的stream中这个值计数，直到stream结尾
  - 计数*c*存在X.val里
- 做多次，对最后的X.val，乘2，减1，乘n，然后求平均 $S = f(X) = n(2 \cdot c - 1)$ $S = 1/k \sum_j^k f(X_j)$



t = 1时采， X.el = a，结束时，有 X.val = m_a

t = 3时采， X.el = b，结束时，有 X.val = m_b

分析



- a
  - 如果在最后一个a采，结束时，有X.val = 1
  - 如果在倒数第二个a采，结束时，有X.val = 2
  - …
  - 如果在t = 1时采，结束时，有X.val = m_a
- 求这些 n(2*X.val - 1) 的均值

$$E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) = \frac{1}{n}\sum_i n\,(1 + 3 + 5 + \cdots + 2m_i - 1)$$

- 因为 $\sum_{i=1}^{m_i}(2i-1) = 2\frac{m_i(m_i+1)}{2} - m_i = (m_i)^2$
- 所以 $E[f(X)] = \frac{1}{n}\sum_i n\,(m_i)^2 = \sum_i (m_i)^2 = S$
- 正是我们要的

推广

- 背后是什么？利用了
- 即 $\sum_{i=1}^{m_i}(2i-1) = 2\frac{m_i(m_i+1)}{2} - m_i = (m_i)^2$

  $-(i+1)^2 - i^2 = 2i - 1$
- 同理，求$(m_i)^3$，就对样本执行

  $c^3 - (c-1)^3 = 3c^2 - 3c + 1$

  再做求和平均
- 推广，求$(m_i)^k$

  $n\,(c^k - (c-1)^k)$

对Infinite Stream

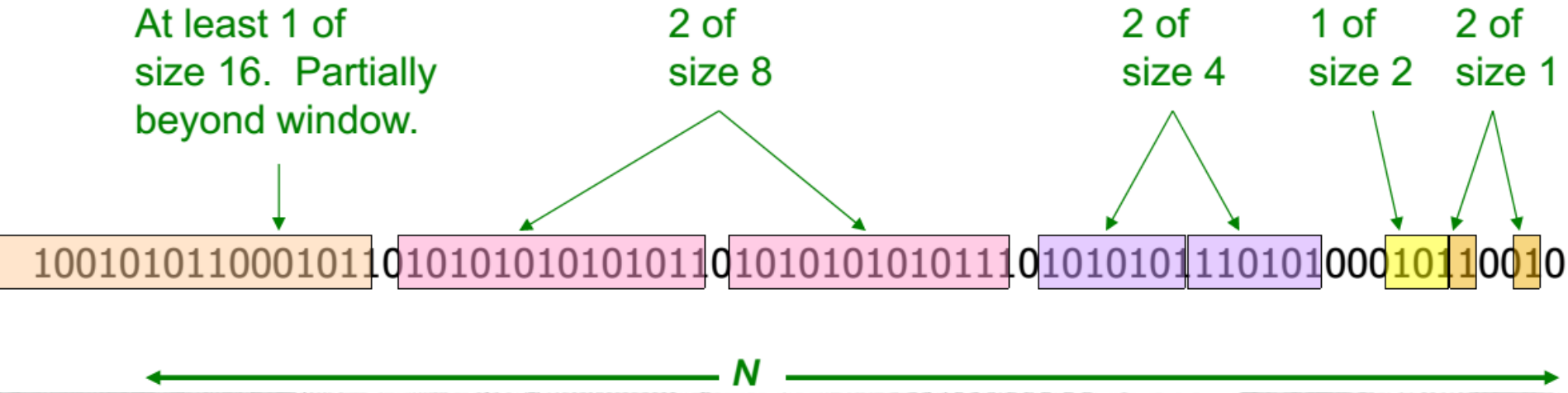
- 采用前面介绍过的固定Size采样办法
  - 采样Size：k
  - 当第n个元素到达时，以k/n的概率留下
- 在采样的*k*个样本中计算*c*
  - 近似得到一个对整个流的矩估计
  - k = 2，Surprise number（奇异数），Popularity分布均匀的度量

发现流行

- 找出过去1个月内，被看次数超过1000的视频？

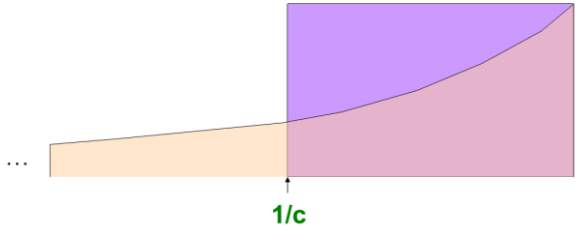
DGIM方法

- 对每个视频，建立一个1/0流，统计1的个数
- 然后挑出超出1000的视频
- 大数据下，太多视频，每个视频一个streaming不现实
  - Youtube，billions of videos



指数衰减窗方法（EDW）

- 启发式方法
  - 我们关心的是“现在”流行啥？
  - 过去的计数，让它们慢慢衰减
  - 热度 = $\sum_{i=1}^{t} a_i(1-c)^{t-i}$
    - a_i：计数
    - c：衰减系数，一般取10^{-6}, 10^{-9}



  - 权重和：$\sum_t (1-c)^t$ is 1/[1 – (1 – c)] = 1/c
- 等价于
  - 来一个新的a，把老热度乘1 - c（即衰减），然后加上这个新a
  - 实现起来非常方便

发现流行

- 实际中，为了减少存储，设一个阈值（如1/2），权重低于该阈值的，就不跟踪了
- 估计要跟踪多少个视频
- 任意时刻，所有视频热度的和
  - 来一个视频观看，以前所有视频观看带来的热度乘（1-c），再给对应视频的热度+1
  - 所有视频观看带来的热度的分布，也是一个等比级数，和为 $\sum_t (1-c)^t$ is 1/[1 – (1 – c)] = 1/c
- 因此，得分超过1/2的电影个数
  - 不会超过2/c
  - 否则，总分将超过1/c
- 所以，最多只需要跟踪2/c个视频的热度
- 省

扩展到一篮子（项集Itemsets）

- 如何用EDW对项集流行度进行跟踪呢？
- 来了一篮子元素B
  - 把所有已有的元素/项集的热度乘 1 – c （衰减）
  - 加新元素
  - 篮子里已有的元素和项集，热度+1
  - 热度 < 1/2的，扔掉
  - B中出现的新子集，如果它的所有子集都在B到来之前被跟踪着，就新增这个子集
    - 直觉：所有子集都热，那它也可能热
  - 例：
    - i,j都在集里了，那么，开始计数$\{i,j\}$
    - $\{i,j\}$ $\{i,k\}$ $\{k,j\}$都在集里了，那么，开始计数$\{i,j,k\}$

跟踪多少个？

- 单item
  - < (2/c) ×篮子里的item数
- 子集
  - 和Load有关

总结