HyperLogLog 简析

林志衡 blog.zhiheng.io
2021/01/23

问题

大数据量去重统计，比如页面 UV。

传统方法

把全部数据加入 Set。

优点：精确
缺点：内存消耗巨大

概率方法

HyperLogLog 算法。

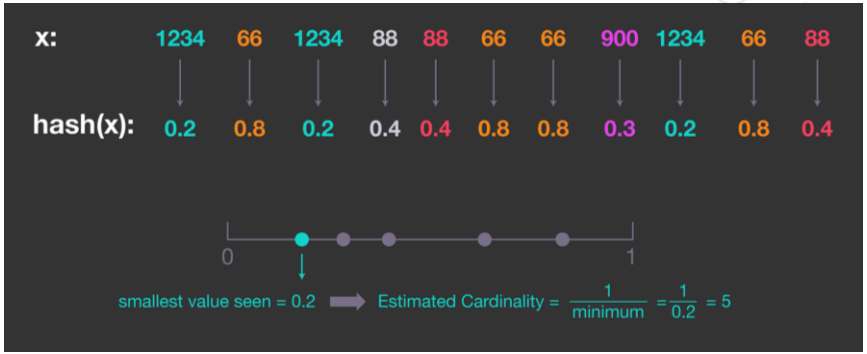
优点：内存消耗很少
缺点：不精确，但误差可以控制在 1% 以内

什么是概率方法?

一个简单例子

将原始数据通过哈希算法随机生成一批均匀分布在 [0, 1) 的数字。

只要随机过程足够均匀，比如图中的 hash(x)。那么：



估算的元素个数 = 1 / 最小的 hash 值

缺点：

如果最小的哈希值碰巧很小，那么估算误差巨大。

另外一种方法

计算末尾连续 0 个数。

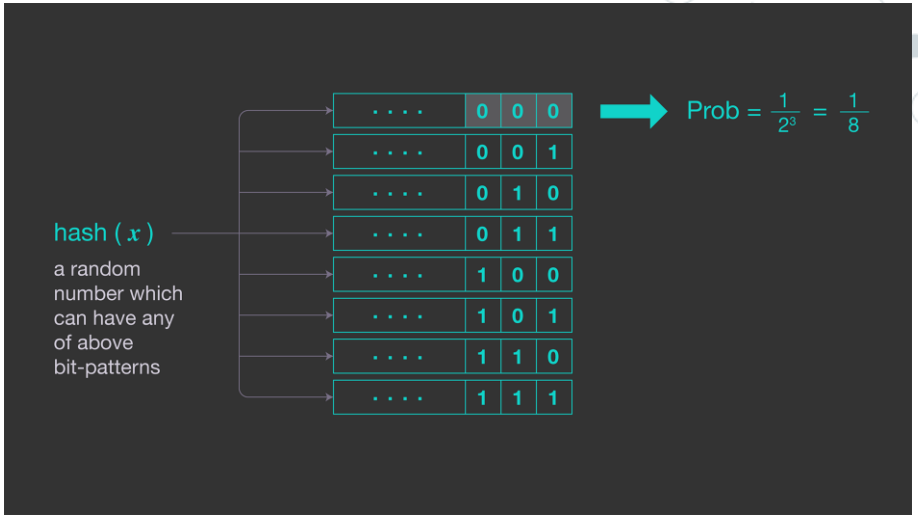
前提：

1. 哈希算法计算出来的值是均匀分布的整数
2. 均匀分布的整数中，末位是连续 n 个 0 的概率为 2^n

结论：

计算一批哈希值中，末位连续 0 最多的数字，比如是连续 R 位。那么：

估算的元素个数 = 2^R



连续末尾 0 计数法

缺点：

1. 估算出来的元素个数 只能是 2 的次方
2. 估算结果可能 不精确（当碰巧出现末尾连续 0 比较多时）

优化精确度

使用多个 hash 函数，生成多批哈希值并估算出多个元素个数值，再做平均值。

优点：

避免单个 hash 函数带来可能的大误差。

缺点：

计算量成倍增加。

LogLog 算法，

1. 取哈希值的前几位作为桶（bucket，也称 register）
2. 按前几位的不同，把数字分入不同的桶中
3. 各个桶分别计算出元素个数，再估算出总数

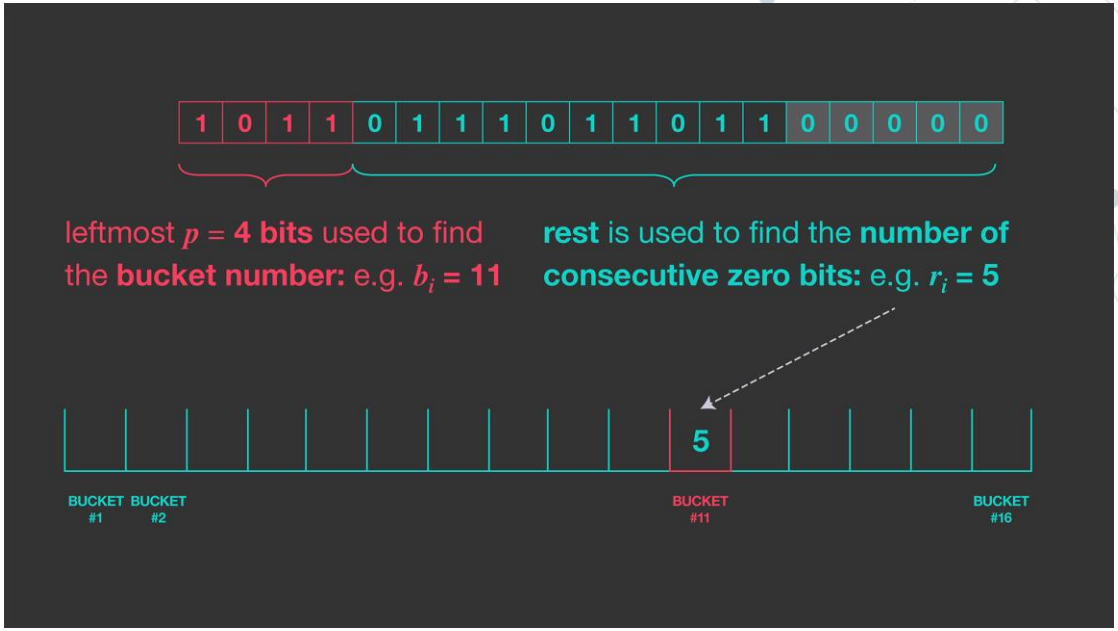
优点：

1. 通过分桶实现与多个 hash 函数一样的效果
2. 计算量不会大量增加

LogLog 算法

比如取前 4 位来生成桶。
有 2^4 即 16 个桶（m 表示）。

数字按其前 4 位分到不同桶。
每个桶可以统计出最长的连续末尾 0 位数，比如图中桶 11 的结果为 5（R_{11} 表示）。



将不同桶的结果取算术平均数。
套入下面的公式：

$$\mathrm{CARDINALITY}_{LogLog} = constant \cdot m \cdot 2^{\frac{1}{m}\sum_{j=1}^{m} \mathrm{R}_j}$$

即可计算出更精确的元素个数。

没有研究数学推导的过程。
constant 取值为 0.79402。
这种方式的误差在 $1.3/\sqrt{m}$。

进一步提升精确度

SuperLogLog：

去掉异常值、使用几何平均数。

精确度提升到 $\frac{1.05}{\sqrt{m}}$。

HyperLogLog：

使用调和平均数。

精确度提升到 $\frac{1.04}{\sqrt{m}}$。

Redis 的 HyperLogLog 实现

- Hash 值为 64 位整数
- 使用前 14 位作为桶（register），一共 16K 个桶（16384）
- 误差在 0.81%

参考

1. HyperLogLog in Presto: A significantly faster way to handle cardinality estimation
   https://engineering.fb.com/2018/12/13/data-infrastructure/hyperloglog/
2. Redis new data structure: the HyperLogLog
   http://antirez.com/news/75

Thanks!

林志衡
blog.zhiheng.io