数据库中基于概率的计数算法

陈嘉杰
计 72 2017011484

2018 年 6 月

摘要

随着时代的发展，数据库中的数据量快速增长，远远超出了单台机器的存储和计算能力，因此，现代的数据库大多采用分布式存储数据的方法，但这也为一些基本的数据库操作添加了困难。本文主要讨论在分布式数据库上，如何实现一个基于概率的计数方法，它对一个存储在分布于不同机器上的一个列表进行计数，得出该列表中相异元素个数的一个估计。由于数据量十分巨大，原本的将所有数据集中在一台机器上进行排序和去重的方案不再现实。但在一些现实需求下，可以牺牲一定的准确性，在较短的时间内得到一个较好的估计。本文对解决该问题的不同算法进行了介绍和评估。

关键词：算法，哈希，数据库，计数算法

1 引言

获取数据库中一个列中所有数据中相异元素的个数在许多地方有应用。通过获取这个数据，可以反映出数据的一些特征，针对该特征，数据库开发者可以根据这个特性进行优化。[1] 在数学上，由于数据库分布式地存储在不同的机器上，这个问题可以转化为：

1. 记我们要统计的所有数据的多重集合为 M 。存储在机器 i 上为多重集合 M 的多重子集 M_i 。
2. 在每台机器上分别对 M_i 进行处理，将处理结果统一到一台机器，在该机器上对 $|M|$ 进行估计，即该多重集合中相异元素的个数 n ，记估计值为 $\hat{n}$。

容易想出解决这个问题的算法，假设只有两个多重子集 A B [1]：

1. 对 A 进行排序，然后对 B 中每一个元素在 A 中进行二分查找，将两个集合中重复的元素去掉
2. 对 A 和 B 分别进行排序，然后进行合并，遇到相同元素时跳过
3. 对 A 和 B 进行哈希等预处理，简化相同元素的比较过程，再采用以上方案

以上这些方案也可以推广到更多的多重子集，但他们的空间复杂度都较高。当数据量极其大，无法在单台机器中容纳时，以上的精确计数方案不在适用。但在允许一定错误率的情况下，通过后文将提到的 HyperLogLog 算法，可以大大减少空间要求，如对一个 10^9 个元素的计数，在 2% 的错误率下也只需要 1.5KB 的内存空间。[2]

本文将介绍几个解决该问题的算法，并对他们进行比较和分析。

2 正文

2.1 前序知识

由于数据本身结构的复杂性，通过直接比较数据是否相同是一个十分低效的过程，并且由于数据本身的特性十分不同，很难设计出一个适用于不同数据类型的普适算法，而且也无法保证数据有一定的概率分布的性质。为此，我们使用哈希函数对数据进行预处理：

定义 2.1.1 (哈希函数) *哈希函数是一个任意数据到 $[0 \cdots 2^L - 1]$ 的映射，其中 L 为哈希的位长，是定值。*

性质 2.1.1 (哈希函数) *哈希函数将任意数据等可能地映射到 $[0 \cdots 2^L - 1]$，每一个二进制位上为 1 和 0 的概率相同*

性质 2.1.2 (哈希函数) *对数据的微小改变，会导致哈希值的巨大变化。反过来说，如果两个数据的哈希值相同，那么几乎可以认为这两个数据相同。*

通过将数据每一个函数进行哈希，我们得到了一个新的多重集合，其中重复的数据，我们可以认为对应着原来数据的重复。因此，统计出新的这个集合的相异元素个数，即得原集合中相异元素的个数。

常见的一些哈希函数如下：

哈希函数	位长（L/bit）
MD5	128
RIPEMD-160	160
SHA-1	160
SHA-256	256
SHA-512	512

通过这一步操作，我们获得了便于处理的二进制数据，也提供了一些概率上的性质。

2.2 Flajolet-Martin 算法 [3]

1985 年，Philippe Flajolet 提出了基于概率的简易计数算法，文中称之为 PC（Probablistic Counting）算法，并通过随机平均（stochastic averaging）提出了改进，为 PCSA（Probablistic Counting with Stochastic Averaging），允许根据需要增加计算量，从而缩小估计值的标准差。

首先定义函数 $\mathrm{bit}(y,k)$ 和 $\rho(y)$ ：

定义 2.2.1 $(\mathrm{bit}(y,k))$ 定义 $\mathrm{bit}(y,k)$ 为 y 的二进制表示中，从右向左第 k 位（从 0 开始）的二进制。

引理 2.2.1

$$y=\sum_{k\geq 0}\mathrm{bit}(y,k)2^k$$

定义 2.2.2 $(\rho(y))$

$$\begin{aligned}\mathrm{rho}(y)&=\min_{k>=0}\mathrm{bit(y,k)}\neq 0 && \text{if}\quad y>0\\ &=L && \text{if}\quad y=0.\end{aligned}$$

可以观察到，由于哈希的均匀分布，我们知道 $\rho(y)$ 为指数分布。对不同的 $\rho(y)$ 进行统计，我们可以得到 PC 算法。

由于哈希函数的分布特性，$BITMAP[0]$ 被更新的期望为 n, $BITMAP[1]$ 被更新的期望为 $n/2$，故我们采用 $BITMAP$ 中最左边一个为 0 的位的位置来估计 $\log_2(n)$ ，并且进行常数的修正，作为 n 的估计值。

我们不加证明地给出，$\sigma(R)\approx 1.12$ 。详细证明见 [3] 。

input : the multiset M whose cardinality is sought
output the estimated $\hat{n}$
:
for $i \leftarrow 0$ **to** $L-1$ **do**
  BITMAP $[i] \leftarrow 0$;
end
for *all* x *in* M **do**
  $index \Leftarrow \rho(\text{hash}(x))$;
  if BITMAP $[index] = 0$ **then**
    BITMAP $[index] \leftarrow 1$;
  end
end
$R \leftarrow$ the position of the leftmost zero in BITMAP;
$\phi \leftarrow 0.77351\cdots$;
$\hat{n} \leftarrow 2^R/\phi$;

Algorithm 1: Probablistic Counting

虽然以上的伪代码中对整体的 M 的每个元素顺序执行，但实践中可以将 M 分拆，在各个机器上分别对 M_i 统计出各自的 $BITMAP$ ，最终，将各个 $BITMAP$ 中的值合并，最后即可得到结果。代码中的 ϕ 由以下等式给出：

$$\phi = 2^{-1/2}e^{\gamma}\frac{2}{3}\prod_{p=1}^{\infty}\left[\frac{(4p+1)(4p+2)}{(4p)(4p+3)}\right]^{(-1)^{\mathrm{v}(p)}} \approx 0.77351\ldots$$

其中 $\mathrm{v}(p)$ 为 p 的二进制表示中 1 的个数。

2.2.1 PCSA（Probablistic Counting Stochastic Averaging）算法

为了进一步缩小标准差，使得我们的估计能够更加接近精确值，通过采用不同的哈希函数，分别求出对应的 $\hat{n}_i$ ，并且取 $\hat{n} = \frac{\sum_{i=1}^{m}\hat{n}_i}{m}$ 为估计值，那么，标准差变为 $\sigma(R) \approx \frac{1.12}{\sqrt{m}}$ 。这样，通过 Stochastic Averaging（随机平均）的方法，我们可以通过改变 m 使得我们可以进一步控制估计的标准差。

2.2.2 评价

这篇论文是这个领域的开山之作，提出了基于哈希的估计算法，这个思想也被后来的许多相似算法提供了基础。它很容易分布式执行，运行速度也很快。但这个算法也有它的局限性：单次运行 PC 算法，所得结果必然为 $2^R/\phi$ 的形式，随机性较大，必须通过多次运行取不同结果的平均值作为最终的估计。

2.3 Linear Counting 算法 [1]

1990 年，Kyu-Young Whang, Brad T. Vander-Zanden 和 Howard M. Taylor 提出了 Linear Counting 算法。这个算法和上面的 Flajolet-Martin 算法有相同之处，它也采用了哈希的算法，不同的是，它采用的是哈希碰撞的次数来对相异元素个数进行最大似然估计。

算法首先创建一个 m 位的位图，然后将每个元素的哈希值对应的位取为 1 。然后，统计 0 的个数，代入公式得到最终的估计。

input : the multiset M whose cardinality is sought
output the estimated $\hat{n}$
:

```
for i ← 0 to m do
    Bitmap [i] ← 0;
end
for all x in M do
    hash_value ← hash(x);
    Bitmap [hash_value%m] ← 1;
end
U_n ← number of "0"s in the Bitmap;
V_n ← U_n/m ;
n̂ ← −m ln V_n ;
```

Algorithm 2: Linear Counting

可以证明 [1]，令 $t = n/m$ ，当 $n, m \to \infty$ 时，有：

$$E(V_n) = e^{-t}$$
$$Var(V_n) = \frac{1}{m}e^{-t}(1-(1+t)e^{-t})$$

容易得出，矩估计 $\hat{n} = -m \ln V_n$ ，并且可以证明这也是最大似然估计。

论文中还提到，如果遇到 Bitmap 填满的情况，这个估计将会有比较大的误差，需要重新选择参数，重复一次以上过程。这就是完整的 Linear Counting 算法。

2.3.1 评价

这个算法采用了和 Flajolet-Martin 算法类似的哈希思路，但得到了来自哈希表的 load factor 启发，设计了这样的算法。它也很容易并行执行，并且最后集中在一处进行最终的估计。局限性在于，为了减小标准差，m 的选取需要从论文中表二进行查表，而不能通过一个简单的公式推算出最优的 m 。而且，由于 m 太小时会出现误差较大的情况，m 不能过小，导致该算法的空间复杂度较高。

2.4 LogLog 算法 [4]

2003 年，Marianne Durand 和 Philippe Flajolet 在原有算法的基础上，提出了 LogLog 算法和它的改进 SuperLogLog 。它同样采用了哈希算法，将数据转为二进制。不同在于，它将数据根据二进制的前 k 位作为划分，将数据分成 2^k 个组，在每个组中分别采用类似于 Flajolet-Martin 算法的估计方案，最后对这 2^k 个估计求出最后的估值。

这里利用了和 Flajolet-Martin 算法一样的性质，区别在于，根据哈希值的前 m 位二进制，把数据分为 2^m 组，每组分别求出对 $\log_2(n)$ 的估计后进行修正。这里的 α_m 由以下等式给出：

$$\alpha_m = (\Gamma(-1/m)\frac{1-2^{1/m}}{\log 2})^{-m}$$

可以证明 [4]，这样得到的无偏估计 $\hat{n}$ 满足：

$$\frac{1}{n}E_n(\hat{n}) = 1 + \theta_{1,n} + \mathrm{o}(1) \qquad \text{where} \quad |\theta_{1,n}| < 10^{-6}$$
$$\frac{1}{n}\sqrt{V_n(\hat{n})} = \frac{\beta_m}{\sqrt{m}} + \theta_{2,m} + 0(1) \qquad \text{where} \quad |\theta_{2,n}| < 10^{-6}$$

input : the multiset M whose cardinality is sought
output the estimated $\hat{n}$
:

let $\rho(y)$ be the rank of first 1-bit from the left in y;
$m \leftarrow 2^k$;
$M^{(1)}, \ldots, M^{(m)} \leftarrow 0$;
for *all x in M* **do**
  $x = b_1 b_2 \cdots \leftarrow \text{hash}(x)$;
  $j \leftarrow (b_1 \cdots b_k)_2$;
  $M^{(j+1)} \leftarrow \max(M^{(j+1)}, \rho(b_{k+1} b_{k+2} \cdots))$;
end
$\hat{n} = \alpha_m m 2^{\frac{1}{m}} \sum_j M^{(j)}$;

Algorithm 3: Basic LogLog

其中 $\beta_\infty = \sqrt{\frac{1}{12}\log^2 2 + \frac{1}{6}\pi^2} \approx 1.29806$ ，由此可得 $\alpha_\infty = \mathrm{e}^{-\gamma}\sqrt{2}/2 \approx 0.39701$

2.4.1 Super Loglog

作者在 LogLog 算法的基础上，提出了改进方案。当得到 $M^{(i)}$ 的结果时，可以选出其中最小的 $\lfloor \theta_0 m \rfloor$ 个元素进行最终的估计，其中 θ_0 为一个 0 到 1 之间的实数，作者推荐采用 0.7。同时，只采用 $[0 \ldots B]$ 中的值，其中 $\lceil \log_2(\frac{N_m ax}{m}) + 3 \rceil$ ，可以减少存储空间的使用。通过这两个优化，算法的空间使用得到优化，并且标准差从 $1.30/\sqrt{m}$ 降为 $1.05/\sqrt{m}$

2.4.2 评价

这个算法很好地解决了 Flajolet-Martin 算法中需要多次采用不同哈希算法取平均的问题，通过将哈希值分成两段，前半段用于分块，后半段用于估计，最后进行整体修正。同时，虽然牺牲了一定的精确度，这个算法的空间复杂度较小，在面对很大的数据量时，拥有较好的优势。

2.5 HyperLogLog 算法 [2]

2007 年，Flajolet 等人在 LogLog 算法的基础上进行改进，提出了 HyperLogLog 的算法，进一步降低了内存占用，同时提高了精确度。算法过程与 LogLog 基本相似，但稍有不同，在最后估计时使用了调和平均。

input : the multiset M whose cardinality is sought
output the estimated $\hat{n}$
:
let $\rho(y)$ be the rank of first 1-bit from the left in y;
$m \leftarrow 2^b$;
$M[1], \ldots, M[m] \leftarrow 0$;
for *all x in M* **do**
  $x = x_1 x_2 \cdots \leftarrow \text{hash}(x)$;
  $j \leftarrow 1 + (x_1 \cdots x_b)_2$;
  $M[j] \leftarrow \max(M[j], \rho(b_{k+1} b_{k+2} \cdots))$;
end
$Z \leftarrow \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$;
$\hat{n} = \alpha_m m^2 Z$;

Algorithm 4: Basic LogLog

其中：

$$\hat{n} = \frac{\alpha_m m^2}{\sum_{i=1}^{m} 2^{-M[j]}}$$

$$\alpha_m = \left(m \int_0^{\infty} \left(\log_2 \frac{2+u}{1+u} \right)^m \mathrm{d}u \right)^{-1}$$

论文中也给出了 α_m 的一些数据和近似计算公式：$\alpha_{16} = 0.673$ ； $\alpha_{32} = 0.697$ ； $\alpha_{64} = 0.709$；$\alpha_m = 0.7213/(1 + 1.079/m)(m \geq 128)$ 。同时，还对不同范围的 n 进行了特殊处理：当范围过小或者范围过大时，需要进一步的修正，最终达到 $1.04/\sqrt{m}$ 的标准差。

2.5.1 HyperLogLog++ 算法 [5]

2013 年，Stefan Heule，Marc Numkesser 和 Alexannder Hall 提出了 HyperLogLog 的进一步改进算法 HyperLogLog++。它直接采用 64 位的哈

希函数，足以解决大多数数据规模；在小数据集上，HyperLogLog++ 添加了一定的启发性，在数据很小的时候采用更精确的 Linear Counting 算法。同时，通过稀疏表示法，进一步减少了空间的消耗。

3 总结

解决这个问题的 Flajolet-Martin 算法从 1985 年提出，经过多次改进和优化，最终得到了现在在数据库中得到广泛应用的 HyperLogLog 算法。HyperLogLog 算法并不是最优的，2010 年，Daniel M Kane 等人提出了解决这个问题的最优解，但由于其实现的复杂性，其广泛性不及 HyperLogLog 算法。[6] 这些算法很好地利用了哈希函数的概率分布性质，将概率统计理论与计算机科学结合，是一个优秀的结合的例子。对于解决这个问题的其它方法，出于篇幅和实践中的使用较少，不予介绍。

4 程序实践

作为一名计算机系的学生，我编程实现了 HyperLogLog 的算法。采用了较近引入 Linux 内核的 eBPF 插桩机制，通过 tc 对进出的网络包进行统计，同时编写了用户态的软件以读出统计结果，并得出相应的估计。项目地址为 https://github.com/jiegec/hll_ebpf 。

通过手动发送目的地址不同的包，可以看到我的程序的输出相应地提升了与包数量相近的数量。这说明我正确地实现了所需要的功能。根据此工具进一步改进，可以实现 DDoS 快速特征检测等特性。

引用文献

[1] Kyu-Young Whang, Brad T Vander Zanden, and Howard M Taylor. “A Linear-Time Probabilistic Counting Algorithm for Database Applications.” In: *ACM Trans. Database Syst.* 15.2 (1990), pp. 208–229.

[2] Philippe Flajolet et al. “Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm”. In: *Conference on Analysis of Algorithms* (2007).

[3] Philippe Flajolet and G Nigel Martin. “Probabilistic counting algorithms for data base applications”. In: *Journal of Computer and System Sciences* 31.2 (1985), pp. 182–209.

[4] Marianne Durand and Philippe Flajolet. “Loglog Counting of Large Cardinalities (Extended Abstract).” In: *ESA* 2832.Chapter 55 (2003), pp. 605–617.

[5] Stefan Heule, Marc Nunkesser, and Alexander Hall. “HyperLogLog in practice - algorithmic engineering of a state of the art cardinality estimation algorithm.” In: *EDBT* (2013), p. 683.

[6] Daniel M Kane, Jelani Nelson, and David P Woodruff. “An optimal algorithm for the distinct elements problem.” In: *PODS* (2010), p. 41.