Computer Science and Application 计算机科学与应用, 2014, 4, 119-124
Published Online July 2014 in Hans. http://www.hanspub.org/journal/csa
http://dx.doi.org/10.12677/csa.2014.47018

Hans汉斯

Comparison of View-Size Estimation Algorithms in OLAP

Xinchen Cui*, Zhenlin Chen, Fang Zhao

Department of Engineer Weapon Science and Technology, Naval Aeronautical and Astronautical University, Yantai
Email: *cuixinchen@sina.com, czlzhy@163.com, 850119362@qq.com

Received: May 22nd, 2014; revised: Jun. 20th, 2014; accepted: Jul. 1st, 2014

Abstract

It must be quick, accurate, and reliable when the size of views are estimated in OLAP. Many methods to deal with view-size estimation apply specific statistical assumptions but their error may usually be large. In comparation, probabilistic techniques have slower speed, but the estimate has higher accuracy and reliability by using less memory. Several hashing-based view-size estimation methods were introduced and analyzed experimentally in this paper. The results showed that the Adaptive Counting provided accurate estimates regardless of the size of view, and its estimated speed remained constantly fast as the memory budget increased.

Keywords

View-Size Estimation, Materialized Views, OLAP, Data Warehouse.

用于OLAP的视图大小估算算法比较与分析

崔欣辰*，陈振林，赵　芳

海军航空工程学院兵器科学与技术系，烟台
Email: *cuixinchen@sina.com, czlzhy@163.com, 850119362@qq.com

收稿日期：2014年5月22日；修回日期：2014年6月20日；录用日期：2014年7月1日

*通讯作者。

摘　要

OLAP系统中的视图物化操作，要求快速、可靠而精确。许多视图大小估算技术利用特定的统计假设，其误差可能较大。基于概率的估算方法在速度方面可能较慢，但是在估算大视图时精确度和可靠度较高，而且使用内存较少。论文中介绍了几种基于散列的视图大小估算方法，并进行了实验加以分析对比。实验结果表明，修正算法(Adaptive Counting)不管视图大小如何均提供精确的估算，而且当增大存储预算时仍可保持较快的估算速度。

关键词

视图大小估算，视图物化，联机分析处理，数据仓库

1. 引言

OLAP 系统中，频繁需要在大规模数据仓库上进行复杂的查询，为了提高查询速度，往往需要事先物化一些视图。视图物化是提高数据仓库性能的最有效技术之一。物化了的视图是一种通过预计算中间结果来提高数据访问的物理结构。进行视图物化首先要解决视图大小的估算问题。

近年来对于视图大小的估算研究很多。典型的 OLAP 查询由选择和聚合数据(通过 GROUP BY 语句实现)构成[1]。通过预计算，许多似是而非的组集可以避免由于大数据表上的聚合引起的响应速度减慢。许多查询，如那些包含条件(HAVING clause)可以利用预先聚合加快计算速度。然而物化视图需要额外的存储空间。而导致在数据仓库刷新时产生维护开销。此外，视图的数据量很大：在一个 d 维的数据立方体中，视图的数量是 $2d$[1]。因此，在数据仓库的物理设计中物化视图的选择是非常重要的，是一个 NP 困难问题[2]。一些视图大小估算技术假设数据的分布是“谦虚”的，通常假设是均匀的分布[3]，但是任何数据分布的偏差均将导致过高的估计结果。

一般的，当统计假设估计量较快完成计算那么代价最大的步骤可能是随机抽样。其误差可能比较大，且不能成为被束缚的先验。这里将讨论基于概率的估算(Probabilistic Counting) [4]、重对数概率估算(LOGLOG Probabilistic Counting) [5]、修正算法(Adaptive Counting) [6]。代价相对大、相对平易的估计量倾向于提供较高的精确度和可靠性[7]。

为了使用这些技术，需要将列进行快速散列(Hashing)，理论边界需要至少两两独立的散列值。幸运的是，当在数据立方体中具有较多维数(d > 10)每一维值相对可用内存来说通常是小的。因此可以对每一维分别进行散列。将结果存储于主存之中，将这些完全独立的一维的散列值合并为独立的 d 型多维散列值。

当分配较多单元时，算法更为精确，但速度减慢。文章中设计了两种使用场合，一种情况是期望粗略估计，误差可达 10%，尽可能的快速估算，这时可使用较小的存储预算(少于 1 MB)；另一种情况是期望精确计算，误差小于 1%，甚至 0.1%，这时使用数 MB 的内存。

2. 多重分型模型的估算

这里在多重分型模型之上实现了各种算法[8]。给定一个样例，对于多重分型模型所有需要研究的是每个样例 F_0 中相异的元素数量，样例 N' 中的元素数量，所有元素的总数 N，以及样例 $m_{\max}$ 中最频繁出现项的数量。因此，一种简单的实现是可能的(见 Algorithm 1)。算法内存的使用由取样上的 GROUP BY 查询决定(line 6)：通常，一个较大的样例将导致更为重要的内存使用。

3. 随机散列

散列以随机的方式进行。从元组$\left[0,2^L\right)$的散列函数中。L 是定值($L=32$ 或 $L=64$)在 $P\left(h\left(x\right)=y\right)=1/2^L$

Algorithm 1. View-size estimation using a multifractal distribution

1: INPUT:Fact table t containing N facts

2: INPUT:GROUP BY query on dimensions $D_1, D_2, \cdots, D_d$

3: INPUT:Sampling ratio $0 < p < 1$

4: OUTPUT:Estimated size of GROUP BY query

5: Choose a sample in t' of size $N' = \lceil pN \rceil$

6: Compute $g = \text{GROUP BY}(t')$

7: let $m_{\max}$ be the number of occuences of the most frequent tuple $x_1, x_2, \cdots, x_d$ in g

8: let F_0 be the number of tuples in g

9: $k \leftarrow \lceil \log F_0 \rceil$

10: while $F < F_0$ do

11: $p \leftarrow (m_{\max}/N')^{1/k}$

12: $F \leftarrow \sum_{a=0}^{k} \binom{k}{a} \left(1 - \left(p^{k-a}(1-p)^a\right)^{N'}\right)$

13: $k \leftarrow k + 1$

14: $p \leftarrow (m_{\max}/N')^{1/k}$

15: RETURN: $F \leftarrow \sum_{a=0}^{k} \binom{k}{a} \left(1 - \left(p^{k-a}(1-p)^a\right)^{N'}\right)$

的前提下，对所有的 x, y，散列是统一的，也就是说所有的散列值是近似平等的。对所有的 x_i, y_i，如果

$$P\left(h(x_1) = y_1 \wedge h(x_2) = y_2\right) = P\left(h(x_1) = y_1\right) P\left(h(x_2) = y_2\right) = 1/4^L$$

那么散列是两两独立的，而两两独立暗示着均匀性。对所有的 x_i, y_i，如果 $P\left(h(x_1) = y_1 \wedge \cdots \wedge h(x_k) = y_k\right) = 1/2^{kL}\ y_i$，则为独立的 k 型智能单元。最后，若对所有的 k 值均为 k 型智能单元，那么散列是完全独立的散列值，需要 $\Omega(F_0)$ 个单元的内存[9]。F_0 很大时将是不切实际的。

k 型独立单元的散列值可在多维数据仓库中高效计算。对每一维 D_i，建立一个查找表 T_i，以 D_i 值作为主键。每当遇到一个新的主键值，生成一个 $[0, 2^L)$ 之间的随机数，并且将其存放在查找表 T_i 中，这个随机数即此键值的散列值。在数据仓库的索引中，维数众多，每一维通常具有几个不同的值，因此，查找表经常至少需要使用几兆的内存。当散列 $D_1 \times D_2 \times \cdots \times D_k$ 上的一个元组 $x_1, x_2, \cdots, x_k$，其散列值为 $T_1(x_1) \oplus T_2(x_2) \oplus \cdots \oplus T_k(x_k)$ 这里 $\oplus$ 为异或。此散列为 k 型智能单元，需要固定的时间完成。表 T_i 可被几个估算多次使用：可同时估算在 D_1 和 D_2 上的 GROUP BY 和 D_2 和 D_3 上的 GROUP BY 的视图大小。但只需一个单独的表 T_2。

4. 算法描述

论文中选用 Probabilistic Counting (基于概率的估算) [4]的算法描述如 Algorithm 2 所示。选用的 LOGLOG Probabilistic Counting (多重对数概率估算)是一个加速的变体[5]。这两种算法的主要区别在于 LOGLOG 算法只跟踪前导零的最大值，而概率算法跟踪前导零的所有观测值。而且对于散列值中的离群值更具弹性。通过实践得到的对于同一参数 M 对这两种算法使用的比较结果如下：概率算法使用 M 个计数器存放从 1 至 $L - \log M$ 的整数。假设独立散列，这两种算法产生相对标准误差分别为 $0.78/\sqrt{M}$ 和

Algorithm 2. View-size estimation using Probabilistic Counting

1: INPUT:Fact table t containing N facts
2: INPUT:GROUP BY query on dimensions $D_1, D_2, \cdots, D_d$
3: INPUT:Memory budget parameter $M = 2^k$
4: INPUT:Independent hash function h from d tuples to $\left[0, 2^L\right)$
5: OUTPUT:Estimateed size of GROUP BY query
6: $b \leftarrow M \times L$ matrix(initialized at zero)
7: for tuple $x \in t$ do
8: $x' \leftarrow \pi_{D_1, D_2, \cdots D_d}(x)$ {projection of the tuple}
9: $y \leftarrow h(x')$ {hash x' to $\left[0, 2^L\right)$}
10: $\alpha = y \bmod M$
11: $i \leftarrow$ position of the first 1-bit in $i \leftarrow \left[y/M\right]$
12: $b_{\alpha,i} \leftarrow 1$
13: A $\leftarrow 0$
14: for $\alpha \in \{0,1,\cdots,M-1\}$ do
15: increment A by the position of the first zero-bit in $b_{\alpha,0}, b_{\alpha,1}, \cdots$
16: RETURN: $M/\phi 2^{A/M}$ where $\phi = 0.77351$

$1.3/\sqrt{M}$。这些理论上的结果假设为独立散列的，实际上是不能实现的，而且要求视图的大小非常大，但是却可以检测小视图。一个小视图相对可用内存 M 来说，可剩下 M 个计数器中的几个不用(Algorithm 2 中的序列 M)。当超过 5%的计数器剩余未用，返回线性计数估算而不是重对数估算[10](见 Algorithm 3 最后一行)。

5. 实验结果

为了进行上述算法的精确度和速度的对比实验，利用测试数据生成程序 DBGEN 生成了规模因素参数等于 2 的测试数据集，大小为 1.5 G。从测试数据集中选择了 8 个视图，这些视图有 2 个视图是 4 维以上的结构。这里标准误差使用对真实值的偏差或由 $\sqrt{E\left(X-c\right)^2}/c$ 确定。这里的 C 是需要估计的值。标准误差(相对)定义为误差的偏离值，通过上式利用 20 次的估算计算所得。

实验运行环境为英特尔双核至强处理器(2.66 GHz)，内存 2G，GUN C++ 编译器 4.0.2。选用 GUN/CGI STL 扩展 hash-map 作为 C++标准集成 unordered_map，提供插入和查询的摊销 $O(1)$。其它的查找表使用 STL map 模版建立，具有红黑树的计算复杂度。所采用的测试内容包括每种估算技术 GROUP BY 查询随机种子和内存的大小、与这些参数相关的每一步计算 GROUP BY 视图的大小及其生成时间。同样的，对多重分型估算技术计算每个 GROUP BY 的时间与估算大小采样率和随机种子。

5.1. 精确度分析

从 DBEGEN 产生的数据集上运行各种算法得到了每种算法的标准误差(见图 1)。

由于 DBGEN 使用一个统一的分布，基于模型的多重分型技术是特别精确的。正因如此，DBEGN 是一个较弱的工具来检测基于模型的视图估算算法。但由于忽略数据分布，文中的这三种算法不存在这种问题。

5.2. 速度分析

在测试数据集上通过上述算法估算所有视图大小所需总时间为 7 min。对多重分型技术，所有的估算

Algorithm 3. View-size estimation using LOGLOG and Adaptive Counting

1: INPUT:Fact table t containing N facts

2: INPUT:GROUP BY query on dimensions $D_1, D_2, \cdots, D_d$

3: INPUT:Memory budget parameter $M = 2^k$

4: INPUT:Independent hash function h from d tuples to $\left[0, 2^L\right)$

5: OUTPUT:Estimateed size of GROUP BY query

6: $M \leftarrow \underbrace{0, 0, \cdots, 0}_{M}$

7: for tuple $x \in t$ do

8: $x' \leftarrow \pi_{D_1, D_2, \cdots D_d}(x)$ {projection of the tuple}

9: $y \leftarrow h(x')$ {hash x' to $\left[0, 2^L\right)$}

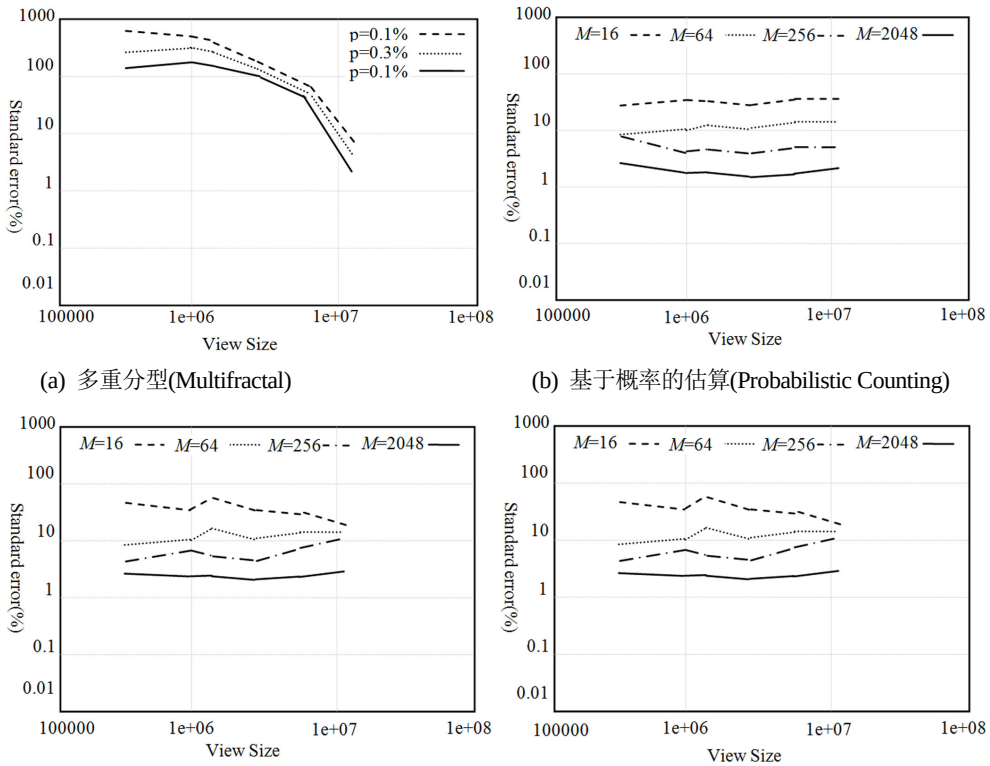
10: $j \leftarrow$ value of the first k bits of y in base 2

11: $z \leftarrow$ position of the first 1-bit in the remaining L-k bits of y(count start at 1)

12: $M_j \leftarrow \max(M_j, z)$

13: (original LOGLOG) RETURN: $\alpha_M M 2^{\frac{1}{M}\sum_j M_j}$ Where $\alpha_M \approx 0.39701 - \left(2\pi^2 + \ln^2 2\right)/(48M)$.

14: (Adaptive Counting) RETURN: $\begin{cases} \alpha_M M 2^{\frac{1}{M}\sum_j M_j} & \text{if } \beta/M \geq 0.051 \\ -M\log \beta/M & \text{otherwise} \end{cases}$ where β is the number of M_j for $j = 1, 2, \cdots, M$ with value zero



(a) 多重分型(Multifractal)　(b) 基于概率的估算(Probabilistic Counting)

(c) 多重对数概率估算(LOGLOG Probabilistic Counting)　(d) 修正算法(Adaptive Counting)

Figure 1. Standard error comparison of view-size estimation algorithms
图 1. 随 M 增大每种估算算法的标准误差分析

Table 1. Running times of the algorithms
表 1. 算法运行时间

	Loading		Hashing		Counting		Time(s)	
	m_1	m_2	m_1	m_2	m_1	m_2	m_1	m_2
(1)	29%	15%	68%	13%	3%	72%	239	240
(2)	30%	13%	70%	11%	1%	76%	235	277
(3)	29%	15%	71%	13%	--	72%	237	240

m_1=256; m_2=8388608; (1): LOGLOG; (2): Probabilistic; (3): Adaptive Counting

约在 2 秒内完成。进一步实验，在一个 5 GB 大小的数据集上，突出每一个处理步骤的用时：装载和解析数据、散列和估算视图的大小(见表 1)。所有算法的运行时间对维数是相当敏感的。对低维(高维)视图，相关更多的时间花费在读取数据上(散列数据)。然而，散列时间和读取时间远大于其余花费在计算的时间。实验结果表明修正算法(Adaptive Counting)不管视图大小如何均提供紧密的估算而且当增大存储预算时，可保持较快的估算速度。

6. 结束语

文章分析了 3 种用于数据仓库环境的视图大小估算算法，通过实验数据得出了各种算法的标准误差，修正算法(Adaptive Counting)提供较为稳定的估算，不受视图大小的影响。存储预算较小时，几种算法的速度相差不大，而增大存储预算时，修正算法仍可保持较快速度。

参考文献 (References)

[1] Faloutsos, C., Matias, Y. and Silberschatz, A. (2012) Modeling skewed distribution using multifractals and the 80-20 law. *VLDB*'12, 307-317.

[2] Gray, J., Bosworth, A., Layman, A. and Pirahes, H. (2013) Data cube: A relational aggregation operator generalizing group-by, crosss-tab, and sub-total. *ICDE*'12, 152-159.

[3] Alon, N., Babai, L. and Itai, A. (2011) A fast and simple randomized parallel algorithms, for the maximal independent set problem. *Journal of Algorithms*, **7**, 567-583.

[4] Whang, K.Y., Vander-Zanden, B.T. and Taylor, H.M. (2013) A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems Online*, **15**, 208-229.

[5] Gupta, H. (2012) Selection of views to materialize in a data warehouse. *ICDT*'12, 98-112.

[6] Golfarelli, M. and Rizzi, S. (2013) A methodological frameworke for data warehouse design. *DOLAP*'13, 3-9.

[7] Flajolet, P. and Martin, G. (2013) Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, **31**, 182-209.

[8] Durand, M. and Flajolet, P. (2012) LogLog counting of large cardinalities. *ESA*'12, Volume 2832 of *LNCS*, 605-617.

[9] Cai, M., Pan, J., Kwok, Y.-K. and Hwang, K. (2005) Fast and accurate traffic martrix measurement using adaptive cardinality counting. *MineNet*'05, 205-206.

[10] Aouiche, K. and Lemire, D. (2007) Unassuming view-size estimation techeniques in OLAP. *ICEIS*'07, 81-95.