第 35 卷 第 5 期 计 算 机 工 程 2009 年 3 月
Vol.35 No.5 Computer Engineering March 2009
·软件技术与数据库· 文章编号：1000—3428(2009)05—0028—03 文献标识码：A 中图分类号：TP311.13

基于 Multi-Bloom Filters 的数据流聚集查询

张 育，沈 鸿
(中国科学技术大学计算机科学与技术系，合肥 230027)

摘 要：针对数据流上任意时间段的历史数据的聚集查询问题，提出基于 BF 技术的概要存储模型 MBF。采用全局比特位向量提供数据元素的快速插入和查找，结合动态分配的局部计数器向量存储不同时间段下的历史数据，使 MBF 支持不同时间粒度上历史数据的有效存储和高效查询，给出历史时间跨度较大情况下 MBF 的压缩方法以及 MBF 模型的参数最优化设置。理论分析证明，MBF 具有较大的灵活性，能有效支持时间范围内历史数据元素的近似聚集查询。

关键词：数据流；历史数据；近似聚集查询；Bloom Filters 技术

Aggregate Queries over Data Streams Based on Multi-Bloom Filters

ZHANG Yu, SHEN Hong
(Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027)

【Abstract】 This paper targets at aggregate queries over historical data in data stream within time intervals, and proposes a novel storage model called Multi-Bloom Filters(MBF) based on Bloom Filters(BF). It uses a global bit vector to realize high efficiency of insertion and query, combines dynamically allocated local counter vectors to store historical data over different time intervals into these counter vectors. Therefore, MBF efficiently supports to store and query historical data over data stream using multiple levels of time granularities. A method of compressing space of MBF is given under the condition that the time span is very large. Analysis shows that MBF has great flexibility and supports approximate aggregate queries over historical data within time intervals, where optimal parameters of MBF are provided in term of query accuracy.

【Key words】 data streams; historical data; approximate aggregate queries; Bloom Filters(BF)

1 概述

目前对数据流的研究集中在分析处理近期数据流数据上，而忽略了对数据流历史数据的分析处理与存储管理。根据数据流的特点，文献[1]从传统的时空数据库技术角度出发，研究了基于 SB-tree 结构的数据流多粒度聚集问题，提出对历史久远的数据采取粗粒度(如以年为单位)的聚集值存储方式，对相对较近的数据采取细粒度(如以分钟为单位)的聚集值存储方式。但这种方法存在的问题是：

(1)对于每个新到达的元素需要及时地更新 SB-tree 结构，并且维护和更新此结构都需要消耗大量的时间，因此，无法满足数据流的实时要求。

(2)SB-tree 结构中的部分节点存在空闲空间，导致存储空间的利用率不高。

(3)不支持分组聚集查询和基于任意时间段的聚集查询。

为了保存原始历史数据，文献[2]提出利用抽样存储实现历史数据的存储，通过初始抽样过程和多层递阶抽样过程减小样本数据对外存空间的存储压力，但是该方法很可能丢失重要的历史数据。

本文结合数据流本身的特性，基于著名的 BF(Bloom Filters)[3]技术，提出了适用于数据流的历史概要存储模型 MBF(Multi-Bloom Filters)。理论分析表明，该结构可以有效地用于数据流历史数据的存储与分析。

2 Bloom Filters

BF 技术通常用于表达庞大的数据集及提高查找效率。其方法如下：设 ***BF*** 是一个比特位向量。首先所有比特位初始化为 0，设有 k 个具有均匀分布特性的哈希函数 $h_1,h_2,\cdots,h_k$ 且 $\forall e\in S$，$h_i(e)\in\{1,2,\cdots,m\}$。对于每个元素 $e\in S$，向量中位置为 $h_1(e), h_2(e),\cdots,h_k(e)$的比特位置 1。查询时，对于 $\forall u\in U$，可以通过观察向量中该元素对应位置的比特位判断：如果其中存在一个或多个比特位值为 0，那么可以肯定 $u\notin S$；反之，以一定错误概率 P_{FP} 得出 $u\in S$，因为 u 所对应的比特位可能被其他元素置 1，且误称概率为

$$P_{FP}=(1-P_0)^k\approx(1-e^{-kn/m})^k$$

BF 技术简单且具有高效的数据表达能力和数据查找效率，但只提供成员关系查询，而且没有考虑元素的删除问题。为了解决 BF 的删除问题，文献[4]提出计数型 BF(CBF)。CBF 使用计数器取代比特位。初始时，所有计数器$(C_1,C_2,\cdots,C_m)$设为 0。当 $\forall e\in S$ 插入 CBF 时，$C_{h_i(e)}=C_{h_i(e)}+1$；当删除集合元素 e 时，$C_{h_i(e)}=C_{h_i(e)}-1$，i=1,2,…,k。这样，CBF 不仅存储了每个元素的频率，还支持删除和更新(先删除，再插入)。查询时，$\forall u\in U$，通过检查 u 对应的计数器的值，返回一个对 u 频率 f_u 的估计：$\hat{f}_u=\min\{C_{h_i(u)}\}$。如果 $\hat{f}_u$=0，可以肯定 $u\notin S$，如果 $\hat{f}_u$>0，可以一定错误概率 P_{FP} 得出 $u\in S$，且 u 在 S 中出现的频率为 $\hat{f}_u$。因为 $f_u\le\hat{f}_u$，所以称 $\hat{f}_u-f_u$ 为误频大小，称 P_{FP} 为 $\hat{f}_u\ne f_u$ 的概率为误频概率，从而推导出[5]：

基金项目：中国科学院“百人计划”基金资助项目
作者简介：张 育(1981－)，男，硕士研究生，主研方向：数据流管理，查询处理和优化；沈 鸿，教授、博士
收稿日期：2008-10-20 **E-mail：**zhangyu110@mail.ustc.edu.cn

$$Pr(f_u \neq \hat{f}) = P_{FF} = P_{FP} \approx (1-e^{-kn/m})^k$$

对于大多数应用，4 位计数器足够保存所对应元素的频率[4]，但是对于数据流这样的多样数据集来说，某些元素的频率可能达到每秒几千甚至几万。此时，4 位计数器将不再适用。文献[5]针对这类应用提出 SBF(Spectral Bloom Filter)，利用变长计数器代替固定长度计数器，并提出相应的索引结构来维持 SBF 的高效查找。但是其结构比较复杂，索引也占用过多的内存空间。为此文献[6]提出了简单又高效的 Dynamic Count Filters，并利用动态计数器向量维护数据的动态增长。

3 MBF

3.1 数据流环境下 Bloom Filters 的缺陷

虽然上述改进扩展了 BF 在数据流方面的应用，但要直接将其应用于数据流历史数据的概要存储还存在很多问题：(1)在给定误差概率下，***BF*** 向量大小是由所存多样集 M 中不同元素的个数确定的，但在很多应用场合下(如数据流)，此信息预先不可知，使得向量大小的确定成为难题。当 $\|M\|$ 估计偏大时，造成存储空间的浪费；当 $\|M\|$ 估计偏小时，误差概率难于保证。(2)单一向量模式建立在无序的多样集合上，这与“数据流是有序的”相矛盾。另外，数据流上历史数据的查询往往是带有时间范围的查询，因此，单一向量模式不支持这样的查询。(3)为了充分利用空间或保证一定的误差概率，可能要将所存时间较久远的元素删除，但是单一向量的 Filters 不提供这种信息，因为所有插入后的数据都没有时间的先后，而且维护这样的信息需要付出很大的代价。

3.2 MBF 数据存储模式

在讨论 MBF 前，先定义其中用到的概念。本文假设时间戳由数据流进入系统的时间确定。由于只考虑离散时间，因此每一个元素的时间戳可以映射成一个整数，从而把一组具有先后顺序的时间戳看作是一个从小到大排列的整数序列。

定义 1 时间段：给定 2 个时间戳 t_s 和 t_e，且 $t_s \leq t_e$，则 $[t_s, t_e]$构成一个时间段(记为 T)，t_s 和 t_e 分别称为 T 的下界和上界，t_e-t_s 的值称为 T 的跨度，记为$|T|$。设有 2 个时间段 T_1= [t_{s1}, t_{e1}], T_2=[t_{s2}, t_{e2}]，如果 t_{s1}= t_{s2} 且 t_{e1}= t_{e2}，则认为 T_1 与 T_2 相等；如果 t_{e1}= t_{s2}，则称 T_3=[t_{s1}, t_{e2}]为 T_1 与 T_2 之和，记为 T_3= T_1+ T_2，并且称 T_1 与 T_2 是可相加的；如果 $t_{s1} \geq t_{e2}$ 或者 $t_{e1} \leq t_{s2}$，则称时间段 T_1 与 T_2 不相交，记为 $T_1 \cap T_2 = \varnothing$；如果 $t_{s1} \leq t_{s2}$，且 $t_{e1} \geq t_{e2}$，则认为 T_1 涵盖了 T_2，记为 $T_1 \square T_2$。

定义 2 数据集的时间段：对于数据流的某个时间段的历史数据集合 S={<d_1,t_1>,<d_2,t_2>,⋯,<d_n,t_n>}，时间戳 $t_s(S)$= min{t_i}, $t_e(S)$=max{t_i}, $1 \leq i \leq n$，$[t_s(S), t_e(S)]$构成数据集 S 的时间段(记为 $T(S)$)。

定义 3 MBF 数据存储模式：如图 1 所示，MBF 具有双层结构，第 1 层由一个大小为 m_1 的全局比特位向量 ***BF*** 构成，第 2 层是由 $i(l \geq i \geq 0)$个大小为 m_2 局部计数器向量(***CBF***$_1$,***CBF***$_2$,⋯,***CBF***$_i$)组成的计数器矩阵 ***CM***，其中分配给每个计数器的比特位个数为 d，且 ***CM*** 的行是可以动态增加的。图 1 中各符号的意义如下：(1) m_1：***BF*** 的大小，即比特位数；(2) k_1：***BF*** 所使用的哈希函数个数；(3) n_1：***BF*** 所能插入的不同元素之个数；(4) m_2：***CBF*** 的大小，即计数器矩阵的列；(5) k_2：***CBF*** 所使用的哈希函数个数；(6) n_2：***CBF*** 所能插入的不同元素的个数；(7) d：分配给 ***CBF*** 中的每个计数器的比特位个数。

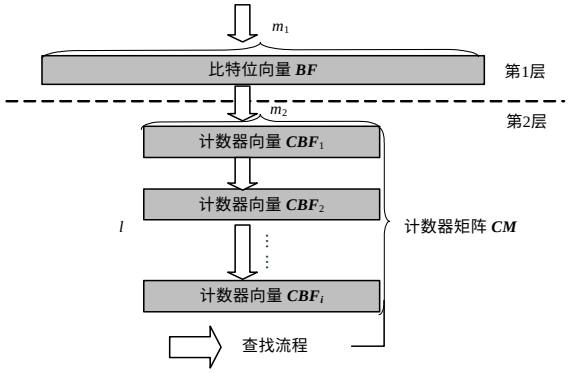

图 1 MBF 数据存储模式

MBF 将数据流划分成多个子窗口(如将数据流中每 1 万个元素为一个子窗口进行划分)，并利用快照模型方法存储数据流中的这些子窗口。本文使用局部 ***CBF*** 存储每个子窗口的概要，另外，为了提高查询效率和降低误差概率，利用全局 ***BF*** 管理一组子窗口(如 10 个连续的子窗口为一组)。算法 1 描述了使用 MBF 的来存储数据流历史数据概要的算法。MBF 第 1 层主要保存元素的成员关系；第 2 层是由与第 1 层相对应的若干个 ***CBF*** 组成，其主要保存元素的频率。每个 ***CBF*** 大小相同并保持相同的 k_2 个哈希函数。因为数据流的无限性，单个 MBF 不能存储整个数据流，因此需要多个 MBF 来满足实际存储的需要，为了提高内存使用率，需要将比较久远的 MBF 放入外存，并通过另外一种数据结构 MBF_Table 来组织和管理这些 MBF。表中每个条目包含的信息如下：标识每个 MBF 的 ID, MBF 所包含数据集时间段，外存地址(如果需要涉及到外存存储 MBF)等。其基本结构如图 2 所示。

MBF$_1$	时间段$_1$	外存地址$_1$	...
MBF$_2$	时间段$_2$	外存地址$_2$	...
⋮	⋮	⋮	⋮
MBF$_{u-1}$	时间段$_{u-1}$	外存地址$_{u-1}$	...
MBF$_u$	时间段$_u$	外存地址$_u$	...

图 2 MBF_Table 的数据结构

算法 1 使用 MBF 保存数据流中的历史元素

输入 数据流 $S = e_1, e_2, \cdots, e_N, \cdots$

Repeat

Create a global BF with size m_1 to store the synopsis of next n_1 elements

Repeat

Create a local CBF with size m_2 for next n_2 elements

For each element e_i in this next n_2 elements

for each bit∈{BF[$h_1(e_i)$], BF[$h_2(e_i)$],⋯,BF[$h_{k2}(e_i)$]} do

BF[$h(e_i)$] = 1

for each counter∈{CBF[$h_1(e_i)$], CBF[$h_2(e_i)$],⋯,CBF[$h_{k2}(e_i)$]}

do

CBF[$h(e_i)$] = CBF[$h(e_i)$] +1

Add the local CBF to the tail of CM

End Repeat

Put the current MBF into external memory and register it in MBF_Table

End Repeat

算法 2 是数据元素基于某个历史时间段计数 *COUNT* 的

查询算法，首先根据查询条件中时间段的跨度从 MBF_Table 中确定所要查询的 MBF。然后对每一个 MBF 判断所要查找的元素是否存在于全局 ***BF*** 中，如不存在，可以肯定该 MBF(或 MBF 所涵盖的历史时间段的数据流)中没有此元素出现；如存在，则查询与全局 ***BF*** 对应的每个 ***CBF*** 中该元素的频率，与 ***CBF*** 一样，通过使用该元素对应的计数器最小值估计在每个局部 ***CBF***(或局部 ***CBF*** 所涵盖的历史时间段的数据流)中此元素出现的频率。最后将涵盖在该查询时间跨度下的所有估计频率进行累加，累加值就是元素基于该历史时间段的计数。

算法 2 查询元素

输入 历史数据元素 e_i 以及查询的时间段

输出 基于查询时间段的元素的计数

```
Look for MBFs from MBF_Table by the input time interval
For each MBF in these MBFs
Probe k1 bits BF[h1 (ei)], BF[h2 (ei)],…,BF[hk1 (ei)]
if none of the above k bits is 0
then
IsInserttion = Ture
Else IsInserttion = false
End if
if IsInserttion
then
    For each CBF in the current MBF
COUNT = COUNT +min{CBF[h1(ei)], CBF[h2(ei)],…,
          CBF[hk2 (ei)]}
Else     COUNT=0
End if
Output COUNT
```

3.3 性能分析

从算法 2 可以看出，查询的误差概率和时间复杂度与查询条件中时间段的跨度密切相关，时间段跨度越大，涵盖的 MBF 个数越多，则查询的时间越长，返回的结果误差也越大。本文只讨论查询单个 MBF 的误差概率和时间效率。

对于任何数据元素，令 P_{FP_BF} 为比特向量的误称概率，P_{FP_CM} 为每个 ***CBF*** 的误称概率，则根据 ***BF*** 的误差概率得

$$P_{FP\_BF}=(1-(1-1/m_1)^{k_1 n_1})^{k_1}$$

$$P_{FP\_CBF}=(1-(1-1/m_2)^{k_2 n_2})^{k_2}$$

那么，整个计数器矩阵的误称概率为

$$P_{FP\_CM}=1-(1-P_{FP\_CBF})^{l}$$

其中，$l(=n_1/n_2)$表示计数器矩阵中计数器向量的个数，因此，整个 MBF 的误称概率为

$$P_{FP\_MBF}=P_{FP\_BF}\times P_{FP\_CM}=P_{FP\_BF}\times(1-(1-P_{FP\_CBF})^{l})$$

为了计算单个 MBF 误频概率，假定 S 为某个 MBF 所表示的多样数据集，$\forall u\in S$，$\hat{f}_u$ 表示从这个 MBF 中查询获得的 u 的频率，f_u 为 u 在 S 中的实际频率。根据查询算法，$f_u\le\hat{f}_u$。令 P_{FF} 为该元素的误频概率，即 $\hat{f}_u\ne f_u$ 的概率，则：

$$P_{FF}=Pr(\hat{f}_u\ne f_u)=Pr(\hat{f}_u<f_u)=1-(1-P_{FP\_CBF})^{l}$$

对于 $\forall u\in U$，根据不同的查询要求，存在不同的查询时间复杂度，如果仅需查询其成员关系，则可以直接从比特位向量中得出查询结果，查找的平均时间复杂度为 $O(k_1)$(此时误称为 P_{FP_BF})，但如果需要查询元素在整个 MBF 中的频率或进一步降低误称概率，则需要在计数器矩阵中做进一步的查询(此时误称概率为 P_{FP_MBF})，此时查找的平均时间复杂度为 $O(k_1+l\times k_2)$。

4 参数设定

由于 MBF 主要用来提供对数据流历史数据的聚集查询，因此从查询的角度考虑，应该根据查询的误差概率和效率的需要给参数设置适当值。

首先根据查询的时间粒度实际需要确定划分数据流子窗口的大小 n_2 和该粒度下的误差概率 P_{FP_CBF}。注意，n_2 取值的好坏将直接影响 MBF 的性能和数据的时间分布效果。如果 n_2 偏大，那么数据的时间分布会很模糊，并且当 n_2 大到能容纳多样集的所有元素时，可以将 MBF 看作是单一向量模式。反之，如果 n_2 偏小，可能导致使用过多的 MBF 来存储多样集，从而降低了查找性能和存储效率。在满足最优 ***BF*** 情况下(当 $k=m\ln 2/n$ 时，Bloom Filters 的误差概率取最小值，即 $PFP=(1/2)^k$)，设置 $k_2=\lfloor\log_{1/2}P_{FP\_CBF}\rfloor$，$m_2=n_2k_2/\ln 2$。这里不考虑计数器溢出的情况，设置 $d=\lceil\mathrm{lb}n_2\rceil$。另外，由于 MBF 中的全局 ***BF*** 主要用来提高查询效率和降低误称概率，因此需要根据这 2 个条件确定 n_2 和 ***BF*** 的误差概率 P_{FP_BF}，同理，设置 $k_1=\lfloor\log_{1/2}P_{FP\_BF}\rfloor$，$m_1=n_1k_1/\ln 2$。

5 压缩 MBF

在许多应用场合下，距离当前较久远的数据其重要性已大大降低，为了充分利用存储空间、提高查询效率，无须将久远的 MBF 与当前 MBF 一样进行细致保存，而是根据实际需要将这些 MBF 进行删除或压缩。删除过程很简单，只需将 MBF 删除并修改其在 MBF_Talbe 中的信息，下面主要介绍如何对 MBF 执行压缩。

本文将压缩过程分为 2 种，第 1 种是对 MBF 内部进行压缩，其主要思想是将多个小的子窗口合并成一个较大的子窗口，如图 3 所示。选择 MBF 内 ***CM*** 中 2 个连续的 ***CBF*** 并将它们对应的计数器相加，为了避免计数器溢出，可能要分配更多的比特位保存累加后的结果。与压缩前的 MBF 相比，压缩后的 MBF 占有的空间约是原来的一半，支持对历史数据粗时间粒度的查询，同时保证了对整个 MBF 的查询误差概率保持不变。

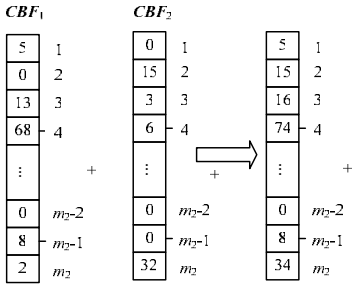

图 3　压缩 MBF 内部的 2 个 *CBF*

第 2 种是在连续的 2 个 MBF 之间进行压缩，其主要思想是将 2 个 MBF 压缩成单个 MBF。首先将这 2 个 MBF 的全局 ***BF*** 合并成单个 ***BF***，如图 4 所示。假设要将 MBF_1, MBF_2 压缩且 MBF_1 所表示的数据比 MBF_2 的久远。方法是将它们的 $\boldsymbol{BF}_1$, $\boldsymbol{BF}_2$ 对应的比特位进行或操作，并将结果保存在 $\boldsymbol{BF}_1$ 中，然后把 MBF_2 里的 ***CBF*** 按照时间顺序加入到 MBF_1 里并修改 MBF_Table。与压缩前相比，压缩后的 MBF 减少了一个 ***BF*** 存储空间，增大了 MBF 的误差概率，但是对历史数据查询的时间粒度保持不变。 （下转第 33 页）