DOI:10.19322/j.cnki.issn.1006-4710.2017.01.003

哈希桶 Variety-B 树的数据流处理方法

王竹荣，伊珍珍，黑新宏，冯华萍，费 蓉

（西安理工大学 计算机科学与工程学院，陕西 西安 710048）

摘要：为方便对数据流数据的存储和查询，分析了 Variety-B 树结构存在的缺陷，设计一种改进的 Variety-B 树结构。通过在内存中开辟循环缓冲区，并在叶子结点采用哈希桶结构，以达到根据流数据信息动态分配内存空间。对哈希桶中数据存储引起的地址冲突设计一种线性探测哈希函数解决方法，及建立 Variety-B 树结构索引，可有效降低地址冲突，提高数据的检索效率。在此基础上，对历史数据流的存储和查询操作算法进行了设计和分析。实验测试结果表明，本文所提改进 Variety-B 树对历史数据流的存储和查询操作所消耗的计算机资源相对 Variety-B 树有所减少。

关键词：数据流；Variety-B 树；循环缓冲区；哈希桶

中图分类号：TP311 **文献标志码**：A **文章编号**：1006-4710(2017)01-0013-05

Storage and query of data stream based on Hash bucket Variety-B Tree

WANG Zhurong, YI Zhenzhen, HEI Xinhong, FENG Huaping, FEI Rong

(School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China)

Abstract: To facilitate the flow of data storage and query, the paper proposes an improved Variety-B Tree structure. By opening up the circular buffer in memory and introducing the concept of hash buckets, a novel hash function is designed to reduce hash bucket address conflicts caused by data storage so as to improve query efficiency. Based on the above methods, the algorithms of store and query of the data stream are designed and analyzed. Experimental results show that the improved Variety-B Tree of computer resources to store and query historical data streams costs less consumption than the Variety-B Tree.

Key words: data stream; Variety-B Tree; circular buffer; Hash bucket

近年来，因为数据流[1]的应用领域越来越广泛，数据流的研究给许多行业带来了显著的经济效益，这使得许多学者对数据流的研究产生了极大的兴趣。通常，传统数据存在磁盘上，以关系型数据的形式呈现出来，数据处理在许多领域有着广泛应用[2]。而数据流不同，它是按照时间顺序快速变化，具有海量和无限延续性等特征，因此很难将其数据全部存储起来。

数据流是指在网络环境中传回的各种类别的监测数据。例如，在线拍卖公司产生的拍卖数据、股票交易所产生的股票价格信息数据、道路交通监测系统和网络监测系统的监测数据、通信运营商即时通信产生的通话数据记录等。由于这些数据具有连续性、突发性、时变性等特征，这使得对数据流的研究较为困难。目前，国外许多知名大学学者对数据流相关问题进行了研究，目前，国外许多学者对数据流相关操作进行了研究，主要工作集中在对数据流的聚集查询[3-4]、存储[5]，以及对数据流增量的操作[6]。

综上，在数据流的操作处理中，存储和查询是两种基本的操作。本文对数据流的研究工作主要包括 2 个方面：(1)在对 B 树[7]研究的基础上，通过引进哈希桶的概念及建立结点索引，设计一种新的 Variety-B 树；(2)将 Variety-B 树用于数据流的存储和查询操作。

收稿日期：2016-05-16

基金项目：国家自然科学基金资助项目(61273127,U1334211)；陕西省重大科技统筹创新资助项目(2015KTZDGY01-04)；陕西省教育厅产业化专项资助项目(15JF024)

作者简介：王竹荣，男，博士，副教授，研究方向为人工智能算法及应用、并行计算、优化。E-mail:wangzhurong@xaut.edu.cn

理论分析和实验测试表明本文对 Variety-B 树结构中引进哈希桶概念，可有效提高数据的存储和查询效率。

1 数据流缓冲区的建立

为模拟数据流处理框架，达到对数据流实时预处理，本文采用缓冲区技术实现对动态数据的分类和存储。

1.1 相关概念

定义 1(数据流) 数据流 S 由连续不断到达缓冲区的流数据组成，它是流数据集合 $S=\{S_1,S_2,S_3,S_4,\ldots,S_i,\ldots\}$，其中 $S_i=(S_{i1},S_{i2},\ldots,S_{in})(i=1,\cdots,n)$ 是 m 维数据，每一维数据表示一种属性，即一条数据流数据由 m 个属性组成。对 $S_j(j=1,\ldots,n)$ 与 $S_k(k=1,\ldots,n)$，若 $j<k$，则 S_j 先于 S_k 到达缓冲区，如果 $k-j=1$，表示 S_j 到达缓冲区后，S_k 紧随其后到达。

定义 2(循环缓冲区) 循环缓冲区是一个环形存储结构，由若干个缓冲单元构成，每一个缓冲单元存储若干条 m 维流数据 $S_i(i=1,\ldots,n)$。假设缓冲区单元空间大小为 M，编号依次为 $0,\ldots,M-1$，$CB_j(j=0,1,\ldots,M-1)$ 表示在循环缓冲区中被存储在编号为 j 的 m 维流数据。

1.2 缓冲区设计及操作

所述循环缓冲区采用一种先进先出的环形队列。它被分配为一块给定大小的内存空间，当数据元素发生变化，仅需改变 head，tail 指针位置，它的结构如图 1 所示。

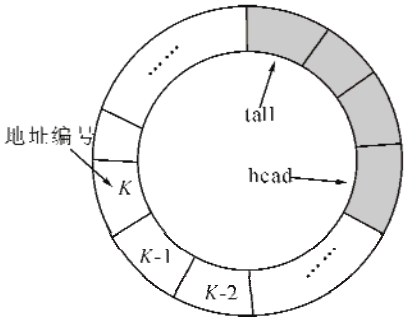

图 1 缓冲区结构

Fig. 1 Buffer structure

1.2.1 缓冲区的工作机制

对图 1 所示的环形队列结构，当执行一次数据插入(push)或弹出(pop)时，head 或 tail 指针会增加 1。因此，当 push 操作将大量数据插入队列时，head 指针沿循环队列到达 tail 指针所在的位置，说明队列已满，不能再进行 push 操作。反之，当多次 pop 操作将数据弹出队列时，会使 tail 追上 head，此时说明队列已空，不能再进行 pop 操作。

1.2.2 数据流的捕捉与处理

循环缓冲区的数据结构主要包含 3 个变量(head，tail 和 elems)，其中 tail 为指向要弹出的元素的索引，head 为指向新插入的元素的索引；另一个是变量向量 elems，它包含若干属性，用于存放缓冲区中的数据。写方法在数据到来时将数据写入缓冲区，修改指针 head。读方法从缓冲区中读数据，修改指针 tail。

1.2.3 缓冲区的主要算法实现

对循环缓冲区 CircularBuffer，缓冲区的最大容量为 MAX_BUFF_LEN。对 Buffer_data 流的一条数据流 f，需要插入到 CircularBuffer，触发缓冲区写操作。该操作首先根据 CircularBuffer 中 head、tail 指针位置关系确定数据插入到 CircularBuffer 的位置，并修改相应指针位置。

2 历史数据流存储结构的设计与实现

在本文中，由于实验数据选用原始股票交易数据，它是一个包含多种类别的数据流集合。将数据流(股票标识、年月日、开盘价、收盘价、成交量、时间戳、涨幅及平均涨幅)等信息组织成一种 Variety-B 树结构，并对 Variety-B 树进行数据流查询等操作。

2.1 Variety-B 树结构

Variety-B 树是 B 树的变形，去掉了 B 树叶子结点指向兄弟的指针，此树的所有结点不能分裂，它是一种层次索引结构。Variety-B 树是一种能存储和索引多维属性[7]数据的内存数据结构。

其结构设计如下：

(1) 根结点、内部结点、叶子结点的键值数据均不相同，其中根结点包含 m 个键值，内部结点包含 n 个键值，叶子结点包含 p 个键值。其中，根结点数据代表一条数据流的分类信息，而叶子结点数据则包含一条数据流的主要特征信息；

(2) 根结点的键值存储在一维结构体数组中，内部结点的键值存储在二维结构体数组中，叶子结点的键值存储在三维结构体指针数组中。

为了支持用户对数据流查询的快速响应，将循环缓冲区输出的数据存储在 Variety-B 树结构中。Variety-B 树的叶子结点包含了全部关键字的信息，这些结点的数据按自小到大的顺序排列。Variety-B 树的结构如图 2 所示。

对图 2 所示 Variety-B 树，各层结点所包含的信息如下：

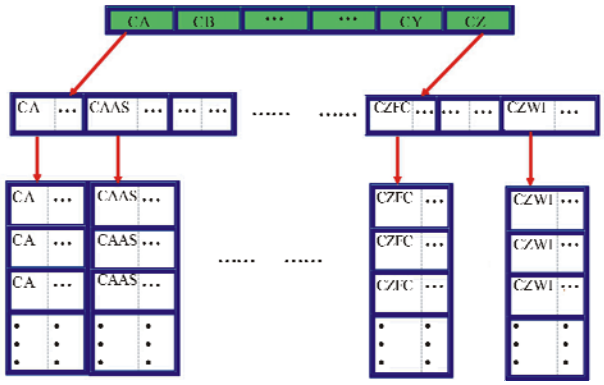

图 2 Variety-B 树结构

Fig. 2 Variety-B Tree structure

A. 根结点的存储信息

根结点的存储信息为（$symbol_1$，$symbol_2$，...，$symbol_{i1}$，... $symbol_m$）。其中 $symbol_{i1}$ 与数据流的标识字段有关，它是一条数据流标识的前缀。本文中，对于数据流信息，将其前缀值取为 CA，CB，...，CZ，作为根结点的 26 个键值。

B. 内部结点的存储信息

内部结点的存储信息为（key_1，key_2，...，key_{i2}，... key_n）。其中 key_{i2} 是关键字，n 由数据流的标识的种类数量决定，key_{i2} 包含 symbol，average_rise，其中 symbol 为某类数据流的标识，average_rise 对应于此种标识的平均涨幅。

C. 叶子结点的存储信息

叶子结点的存储信息为（$keyleaf_1$，$keyleaf_2$，...，$keyleaf_{i3}$，...，$keyleaf_p$）。其中 $keyleaf_{i3}$ 是关键字，它由 symbol，year，month，day，openprice，closeprice，volume，rise，timestamp 等字段信息组合而成，按时间戳（timestamp）由小到大依次有序排列。

对 Variety-B 树的操作包括插入数据流生成一棵 Variety-B 树，及对其进行检索、修改、删除。修改和删除均在给定一组键属性，在 Variety-B 树中检索出该条记录，然后对该记录进行修改或者删除，删除该记录后，与该记录同在一个叶子结点中的其它记录相继前移。

2.2 改进的 Variety-B 树

针对上述 Variety-B 树结构对叶子结点查找效率较低及叶子结点不能分裂等缺点，本文采用哈希算法进行改进。它的基本思想是采用哈希桶存储叶子结点数据，将叶子结点按一定的方式划分分布存储到若干个哈希桶中。每个桶按生成的关键字序列排序。这样处理不仅可提高查询效率，而且也易于 Variety-B 树的横向扩展。

2.2.1 哈希算法的设计过程

为每个叶子结点建立哈希表并开辟一块内存空间，这段内存空间称为“哈希桶”。为每个叶子结点建立哈希表，哈希表中只存储编号，将结点中的数据（几种不同属性，也称关键字集合）通过某种函数关系 f 唯一映射到哈希桶中。当要查找某条记录的时候，就能根据该条记录通过哈希函数生成的编号来确定它在哈希桶中的位置。此外，设定好哈希桶的大小后，内部结点的一个关键字可以对应一至多个哈希桶，即一个类别下的流数据可以被分散至若干个桶中，这样可保证 Variety-B 树具有可分裂性。

本文中，哈希函数定义为以键属性组合 K（symbol、year、month、day 的属性组）为自变量，通过构造一个自定义的确定函数 f，计算出对应的函数值 $f(K)$，$f(K)$ 即为键属性组合 K 的记录在哈希桶中的存储地址。函数 f 的定义如下：

$$f(K)=\text{sum(sy)}*a_1+(\text{year}*100+\text{month}*10+\text{day})*a_2$$

式中，sum(sy) 代表 symbol 的各位字符（sy）ASCII 值之和，a_1 和 a_2 按经验取奇素数，本文取 $a_1=31$，$a_2=17$。

通过上述哈希函数方法可以得到记录的哈希值，该值代表该记录在哈希桶的存放地址，从而提高查询效率。对哈希函数计算导致的地址冲突，本文采用线性探测解决哈希冲突问题，详情将在哈希桶插入和哈希桶查询算法中描述。

2.2.2 哈希桶的操作

哈希桶可进行的操作包括插入、查询、修改和删除。由于修改和删除依赖于查询，要对桶中某条记录进行修改和删除必须先进行查询，查询出来该条记录后，直接对其修改或者删除，相应地更新哈希桶。考虑上述操作之间的相关性，以下主要对哈希桶的插入操作和算法进行分析。

为方便哈希桶的操作，引入辅助数组 hash_fuzhu 存储数据流标识对应的起始哈希桶编号，结构体数组 hs 为哈希桶，整形常量 MAX_HASH 为哈希桶的容量，数组 HASH 存放哈希函数值，整型变量 end 为哈希桶内指向新元素的索引，整型变量 hash_bucket_count 记录哈希桶中元素的个数，初始化哈希桶后 end 为 0，strcat(f. symbol，f. year，f. month，f. day) 为 f. symbol，f. year，f. month，f. day 四个属性变量的串连接值，其它符号同上。

hash_bucket 结构体定义：

```
struct Hash_Bucket{
int end;//指向桶新元素的索引
```

Hash_ElemType * hash_elems;

int count;//哈希桶记录数};

当将 f 插入哈希桶中时,触发哈希桶插入算法,哈希桶插入算法描述如下:

Algorithm 1 (哈希桶插入算法)

Input:root_symbol,sym,f. bucket_no

Output: 1 棵叶子结点采用 Hash Bucket 结构的 Variety-B 树

1. Begin
2. Initialize;
3. Compute Hash_vaule: $f(K)=m*a_1+$ (year * 100+month * 10+day) * a_2;
4. Generate string str←Strcat(f. symbol, f. year, f. month, f. day);
5. for each i[0,MAX_ROOT_NUM) DO
6. Check key symbol position of root
7. end for
8. for each j[0,MAX_INTER_NUM) DO
9. Check symbol position of inter node;
10. end for
11. Compute hash_address: k←$f(K)$%MAX_HASH;
12. if (hash_bucket is full)
13. Generate New hash bucket;
14. end if
15. if(hs[bucket_no1][k]==0) //无冲突
16. hs[bucket_no1][k]=1;
17. Save data info;
18. hash_bucket. count++;
19. else//有冲突
20. Checknewaddress (&k1);
21. hs[bucket_no1][k1]=1;
22. Save data info;
23. hash_bucket. count++;
24. end if
25. End.

当输入 symbol,year,month,day 值进行查询时,触发哈希桶查询操作。考虑该算法过程较为简单,不再详述。

3 实验结果及分析

实验环境:CPU 为 Intel(R) Core(TM)2 Duo 2.8 GHZ,内存为 2 GB,操作系统为 Windows 7。数据流预处理阶段,在 VC++6.0 环境下,采用 C 语言开辟循环缓冲区,模拟数据流处理系统,对数据流进行处理;存储阶段,采用 C 语言设计 Variety-B 树结构。实验中所采用的数据来自国外真实的股票交易历史数据,数据容量为 30 553 kB。下面针对本文所提方法给出几组测试结果。

3.1 运行时间效率

本文给出 Variety-B 树和改进后的 Variety-B 树的查找、生成和修改所消耗时间的对比。从图 3 可看出,在数据规模一定的情况下,改进后的 Variety-B 树相比于 Variety-B 树查询速度快,原因在于哈希算法将原始属性字段映射成整型值,并将此整型值存放在哈希表中,因此索引时速度较快。从图 4 可看出,改进后的 Variety-B 树相比于 Variety-B 树生成所耗时间更少,原因在于改进后的 Variety-B 树所采用的哈希桶具有横向扩展能力,哈希桶结构采用一维数组,直接存储叶子结点信息,因此生成耗时相对较少;而 Variety-B 树叶子结点采用三维指针数组,其中第一、二维都用来作为第三维存储数据信息的索引,因此生成耗时相对较多。

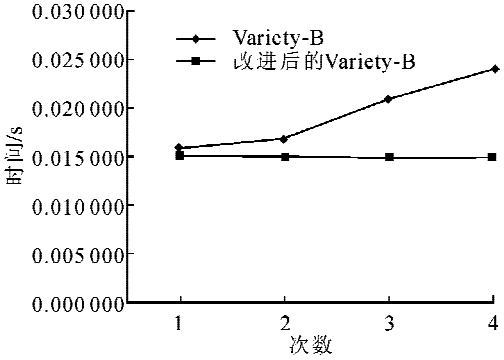

图 3 Variety-B 树和改进后的 Variety-B 树查找时间对比曲线

Fig. 3 Comparative curve of query time between Variety-B Tree and the improved Variety-B Tree

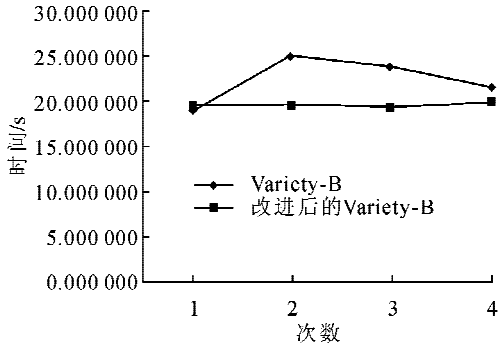

图 4 Variety-B 树和改进后的 Variety-B 树生成时间对比曲线

Fig. 4 Comparative curve of generation time between Variety-B Tree and the improved Variety-B Tree

3.2 内存消耗

实验给出同等规模的数据流加载在内存数据库中，Variety-B 树和改进后的 Variety-B 树所消耗内存空间的对比，从图 5 可以看出，改进后的 Variety-B 树消耗内存空间明显小于 Variety-B 树消耗内存空间，原因在于 Variety-B 树和改进后的 Variety-B 树的根结点和内部结点结构一致，因此两种树结构存储数据在根结点和内部结点所耗内存空间相同；Variety-B 树的叶子结点采用三维指针数组，其中第一、二维都用来作为索引，分别用来定位根结点和内部结点，不同类别的数据分别存储在不同叶子结点上，因为分配给每个叶子结点的内存空间大小相同，而每个叶子结点实际存储的数据量可能不同，有些类别下所包含的数据量较大，有些类别下包含的数据量较小，无论某种类别包含数据多少，均要单独占用一个叶子结点。此外，由于叶子结点不能分裂，部分叶子结点未满，导致一些内存空间被浪费。而改进后的 Variety-B 树所采用的哈希桶结构为一维数组，直接存储叶子结点信息，不同类别的数据可存储在同一个哈希桶中。采用这种处理，除最后一个哈希桶可能未满，其余哈希桶均为满，内存空间被最大限度利用，因此占用内存空间较小。

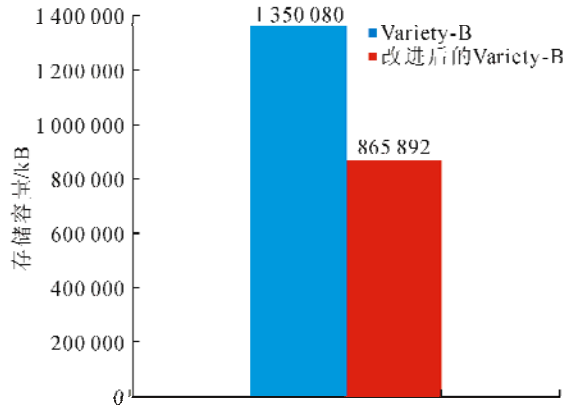

图 5 Variety-B 树和改进的 Variety-B 树所占内存的对比柱图

Fig. 5 Comparative column chart of the running memory between Variety-B Tree and the improved Variety-B Tree

4 总 结

在数据流研究领域，对历史数据流的处理、存储与查询一直都是研究的热点，通过分析与测试，可以看到，本文所提出的 Variety-B 树结构内存数据库可以对循环缓冲区处理的数据进行高效地存储和快速地查询。

本文研究仍有以下几点不足之处：对哈希函数的构造方法进行研究，以便更有效地处理地址冲突；对 Variety-B 树结构做进一步改进使其支持多属性的范围查询，以及对数据流的预测。

参考文献

[1] 郭龙江，李建中，王伟平，等. 数据流上的连续预测聚集查询[J]. 计算机研究与发展，2004，41(10)：1690-1695.
GUO Longjiang, LI Jianzhong, WANG Weiping, et al. Predictive continuous aggregate queries over data streams[J]. Journal of Computer Research and Development, 2004, 41(10): 1690-1695.

[2] 陈浩，华灯鑫，张毅坤，等. 一种基于变换矩阵的激光雷达数据无量纲统一化转换方法[J]. 西安理工大学学报. 2014，30(1)：1-8.
CHEN Hao, HUA Dengxin, ZHANG Yikun, et al. A conversion method for lidar data non-dimensionalization and standardization based on transform matrix[J]. Journal of Xi'an University of Technology, 2014, 30(1): 1-8.

[3] ANANTHAKRISHNA R, DAS A, GEHRKE J. Efficient approximation of correlated sums on data streams [J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(3): 569-572.

[4] 田海生，陈立军. 基于大纲的数据流自适应聚集算子的实现[J]. 计算机应用，2007，27(10)：2383-2387.
TIAN Haisheng, CHEN Lijun. Inplementation of adaptive aggregate operator based on synopsis[J]. Computer Applications, 2007, 27(10): 2383-2387.

[5] 张冬冬，李建中，王伟平，等. 数据流历史数据的存储与聚集查询处理算法[J]. 软件学报，2005，16(12)：2089-2098.
ZHANG Dongdong, LI Jianzhong, WANG Weiping, et al. Algorithms for storing and aggregating historical streaming data[J]. Journal of Software, 2005, 16(12): 2089-2098.

[6] 李菲菲，李红燕，曲强，等. SPQ：数据流上面向可伸缩模式的查询方法[J]. 计算机学报，2010，33(8)：1481-1491.
LI Feifei LI Hongyan QU Qiang, et al. SPQ: a scalable pattern query method over data streams[J]. Chinese Journal of Computer, 2010, 33(8): 1481-1491.

[7] JIN Xing, KEN YIU W P, GARY CHAN S H. Supporting multiple-keyword search in a Hybrid structured peer-to-peer network [C]//2006 IEEE International Conference on Communications, Istanbul, 2006: 42-47.

（责任编辑 杨小丽）