计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
50(3): 594-601, 2013

适用于范围查询的列存储数据桶划分算法

李晔锋　乐嘉锦　王　梅
(东华大学计算机科学与技术学院　上海　201620)
(superbee@mail.dhu.edu.cn)

A Column-Store Based Bucket Partition Algorithm for Range Queries

Li Yefeng, Le Jiajin, and Wang Mei
(*College of Computer Science and Technology, Donghua University, Shanghai* 201620)

Abstract　Range query is significant to databases. In a column-store database, using range queries on attribute values to obtain the resulting row-id set, would affect the performance of tuple reconstruction. Compared with tree structure, Hash tables are more effective in exact queries but less effective in range queries. With this situation, a bucket partition algorithm for range queries is proposed. Firstly, In order to give a good introduction to the algorithm, a Hash storage model used for range queries (ranged hash, RH) is proposed, along with the definition of the bucket range and the serialization. Then, according to the "read-optimized" feature of column store databases, an improved bucket partition algorithm used for range queries is proposed based on the RH model. The algorithm could generate serializable Hash functions to partition attribute values into buckets, and could improve not only the efficiency of range queries but also the storage efficiency. Finally, the experimental results prove the efficiency of the algorithm.

Key words　column-store; range query; Hash table; serializable; bucket partition

摘　要　范围查询是数据库中一项重要的操作.列存储数据库中,能否有效查找一个范围内的属性值,获取对应的行号集合,将极大影响元组重构的效率.与树型结构相比,Hash 表对数据的精确查找具有更高的效率,但是范围查找的效率比较低.针对这种情况,提出了一种改进的可用于范围查询的数据桶划分算法.为了能够更好地对算法进行描述,首先提出了可用于范围查询的 Hash 存储模型(ranged Hash, RH),并给出了桶的值域和序列化的定义.其次针对列存储等"读优先"特性,在 RH 模型的基础上,提出一种改进的桶划分算法.该算法生成可序列化的哈希函数把属性值划分到桶中,能够同时提高属性值的范围查询效率和存储效率.最后,通过实验结果验证算法的有效性.

关键词　列存储;范围查询;Hash 表;可序列化;桶划分

中图法分类号　TP311.1

列存储把属于同一列的属性值连续地、压缩地、密集地存放在磁盘中的同一块区域,当有查询请求到来时,可以避免扫描查询无关的列,提高查询效率[1].在过去的 20 年里,列存储的研究在数据库领域产生了巨大的影响,其中最具有代表性的是 Monet DB[2],C-Store[3]和 Sybase IQ[4]等.

在列存储数据库中,元组重构是一项重要的研究课题,其目的是把逻辑上属于同一条记录的不同

收稿日期:2011-06-22;**修回日期**:2011-12-26
基金项目:"核高基"重大科技专项基金项目(2010ZX01042-001-003-004);国家自然科学基金项目(61070031,61070032);上海市自然科学基金项目(11ZR1401200)

列的属性值重新组装成行[5]. 为了实现这个目的,通常需要满足以下的条件. 首先,为每个属性值分配一个逻辑编号,与属性值构成二元组存储在磁盘上,例如 Monet DB 使用行号(rowid)作为逻辑编号,这样行号相同的属性值可以构建成元组. 其次,每一列对应的属性值可能存在一个或者多个副本,不同副本的逻辑编号具有不同的含义. 例如 C-Store 在创建投影时,可能某一列同时存在多个投影中. 假设表 T 有 5 列,其中列 ABC 构成一组投影,并且按照列 C 的属性值有序排列;ADE 构成另一组投影,按照列 D 的属性值有序排列,则存在列 A 的两个副本,并且各自的逻辑编号包含的意义完全不同. 最后,需要一种存储结构把这些由逻辑编号和属性值构成的数据项组织起来,实现快速地查找. Sybase IQ 支持 B^+ 树索引[6]和位图索引[7],C-Store 的底层采用了 Berkeley DB[8]的结构,它支持 B^+ 树索引和 Hash 表[9]的结构. Berkeley DB 使用线性 Hash[10]的方法对关键字构建 Hash 表,实现数据的快速查询.

在本文中,数据项由行号和属性值组成. 在这个前提下,以如图 1 所示的查询为例,系统会首先查询 *SALARY* 列中属性值为 10 000 的数据项对应的行号值,然后根据该行号分别到 *ENAME* 和 *SEX* 列中找出对应的属性值,最后根据结果重构成所需要的元组. 为了实现行号和属性值的快速查找,通常使用 B^+ 树和 Hash 表组织数据.

```
SELECT ENAME, SEX, SALARY
FROM EMP
WHERE SALARY=10000;
```

Fig. 1 A simple query.

图 1 一个简单查询

若使用 B^+ 树的方式组织数据,原始数据项将按照行号有序存放到各个数据页中,然后根据行号为关键字创建 B^+ 树索引,同样它的副本将创建以属性值为关键字的 B^+ 树索引. 若使用 Hash 表的方法组织数据,需要分别以行号和属性值为关键字,为原始数据和副本各选择一个 Hash 函数,然后把数据项依次存放到桶中.

对于图 1 所示的查询,B^+ 树需要通过逐层遍历才能获得属性值对应的数据项所有的数据页,而 Hash 表能够通过 Hash 函数立即获得数据项在的桶,显然使用 Hash 表的性能要高于 B^+ 树索引. 如果把 WHERE 子句中的等号改成大于或者小于,原先的等值查询变成了范围查询. 由于 B^+ 树的叶子结点是有序的,它仍然支持这类范围查询. 然而 Hash 表中的数据是离散的,为了能够查询所有满足条件的数据,系统不得不对 Hash 表中的每个桶进行遍历,此时使用 Hash 表的性能将远远低于 B^+ 树. 由此可见,有序性对于范围查询非常重要. 此时,一个直观的解决方案是先对数据项按照属性值排序,然后存放到桶中. 当数据量巨大时,内部排序无法单独完成属性值的排序,通常需要借助外部存储设备. 因此使用排序的方法创建属性值副本将导致额外的时间和存储空间的开销.

针对上述问题,本文提出了一种改进的桶划分算法,使得 Hash 表中桶的分布具有一定的规律,从而提高属性值的范围查询效率及存储效率. 本文主要贡献如下:1)通过定义逻辑编号、物理编号、序列化等概念,明确给出了可用于范围查询的 Hash 存储模型(ranged Hash, RH);2)针对列存储等"读优先"系统,提出一种改进的桶划分算法,清晰地给出可序列化的 Hash 函数定义,利用该 Hash 函数进行 Hash,不仅能够满足属性值的范围查询效率并且有效提高存储效率.

1 线性 Hash

1.1 线性 Hash 定义

线性 Hash 由 Litwin[10]首次提出,它的作用在于减少溢出桶的个数,在创建 Hash 表的过程中,Hash 函数动态地发生改变,同时对相应的桶执行分裂操作.

Litwin 对动态 Hash 函数描述如下:

设 C 表示关键字集合,$h_0:C\rightarrow\{0,1,\cdots,N-1\}$ 为初始 Hash 函数,$h_1,h_2,\cdots,h_i,\cdots$ 称为 h_0 的分裂函数[11],它们具有以下的特点:

$$h_i:C\rightarrow\{0,1,\cdots,2^iN-1\},\tag{1}$$

$$\forall c\in C, h_i(c)=h_{i-1}(c) \text{ 或}$$

$$h_i(c)=h_{i-1}(c)+2^{i-1}N.\tag{2}$$

假设初始 Hash 函数为 $h_0:c\rightarrow c \bmod N$,对于任意的非负整数 K 和 L 均满足:

$$K \bmod 2L = K \bmod L$$

或

$$K \bmod 2L = K \bmod L + L.$$

根据上面两式,可以推出分裂函数的表示形式为

$$h_i:c\mapsto c \bmod 2^iN.\tag{3}$$

1.2 桶的分裂

Litwin 指出,线性 Hash 使用了延迟分裂的方法. 假设 n 表示一个指针,指向即将执行分裂操作的

桶,初值为 0. 当某个桶中发生冲突时,先产生一个溢出桶,直到装载因子 $\hat{a}$ 达到某个阈值时,才会对 n 指向的桶执行分裂操作,并且 n 指向下一个桶. 装载因子的值由记录的总数 x、主桶的容量 b、溢出桶的容量 b'、主桶的个数 M 和溢出桶的个数 M' 决定:

$$\hat{a}=\frac{x}{bM+b'M'}. \tag{4}$$

1.3 桶的定位

要查找某一个值 c,需要获得桶的编号 m. 文献[10]对桶的编号给出了以下的命题:

引理 1. 对给定的关键字 c,桶的编号 m 与指针 n 的值无关,它由主桶个数 M 和分裂周期 i 决定,即

$$m=\begin{cases}h_{i-1}(c), & \text{若 } h_i(c)>M;\\ h_i(c), & \text{若 } h_i(c)\in[0,M].\end{cases} \tag{5}$$

Litwin 对该命题给出了详细的证明,该命题同样适用于向 Hash 表中追加数据时确定桶的编号.

2 RH 模型

2.1 相关定义

设 R 表示行号集合,V 表示表中某一列属性值的集合,则 $D=\{(r,v)\mid r\in R, v\in V\}$ 表示构成该列所有数据项的集合,$B_m=\{(r_{mj},v_{mj})\mid m\in\mathbb{N}, j\in[1,b]\}$ 表示第 m 个桶中所有的数据项,其中桶的编号为 m,每个桶的最大容量为 b,显然 $\sum_m B_m=D$. 若分别选取 r 和 v 作为关键字,则数据项集合 D 具有两种桶划分方式,每种划分方式分别使用不同的 Hash 函数 H_r 和 H_v. 其中 H_r 用于存储原始数据,可以根据行号查询属性值;H_v 用于存储副本,可以根据属性值查询行号.

定义 1. 桶的值域. 对任意的 $(r_{mj},v_{mj})\in B_m$, $j\in[1,b]$,当选取行号作为关键字时,存在一个最大值 $r_{\max}$ 和一个最小值 $r_{\min}$,则该桶关于行号的值域表示为:$dom_r(B_m)=[r_{\min},r_{\max}]$;当选取属性值作为关键字时,存在一个最大值 $v_{\max}$ 和一个最小值 $v_{\min}$,则该桶关于属性值的值域表示为:$dom_v(B_m)=[v_{\min},v_{\max}]$. 如果没有特别指定关键字,则该桶的值域表示为 $dom(B_m)$,符号 sup 和 inf 分别表示值域的上界和下界.

若 $dom(B_m)\cap dom(B_{m'})=\varnothing$,并且满足 $\inf(dom(B_m))>\sup(dom(B_{m'}))$,则称 $dom(B_m)>dom(B_{m'})$,表示 m 号桶中的任意一个关键字都大于 m' 号桶中的关键字,反之亦然.

定义 2. 序列化. 设 Hash 函数 H 生成的桶个数为 M,若对于任意的自然数 $p,q\in[0,M)$,都有 $dom(B_p)\cap dom(B_q)=\varnothing$,则称 H 为可序列化的 Hash 函数;若对于任意的 $p>q\in[0,M)$,$dom(B_p)>dom(B_q)$,则称 Hash 函数 H 为序列化的 Hash 函数. 若存在可逆函数 $G\in[0,M')$,$M'\geqslant M$,桶的编号 $m\in[0,M)$ 为其自变量,使得 $\forall g_1>g_2\in G$,$dom(B_{G^{-1}(g_1)})>dom(B_{G^{-1}(g_2)})$,则称函数 G 为序列化函数,其函数值称为桶的逻辑编号,对应的自变量 m 称为桶的物理编号,使用函数 G 把物理编号转化成逻辑编号的过程,称为序列化.

为了实现 Hash 表的范围查询,Hash 函数必须至少是可序列化的,因为序列化保证随着桶的逻辑编号递增,对应桶的值域中的上下界也递增,这样在执行一个范围查询时,可以通过 Hash 函数和序列化函数快速定位到桶逻辑编号的上界和下界,从而确定需要遍历的桶物理编号的范围.

定义 3. 可范围查询的 Hash 表. 由可序列化的 Hash 函数或序列化的 Hash 函数生成的 Hash 表,称为可范围查询的 Hash 表(ranged Hash table, RHT).

2.2 基于行号和属性值的桶划分模型

当选取 r 作为关键字时,由于 r 值唯一,并且随着元组的插入依次递增,在桶中也是按照先后顺序存放. 因此可以构建序列化的 Hash 函数 H_r,使得:

$$\forall r_p>r_q\in R, H_r(r_p)\geqslant H_r(r_q)(p\neq q), \tag{6}$$

其中当 r_p 和 r_q 在同一个桶中时,Hash 值相等.

H_r 的构建比较容易,由于每个数据项都是依次有序地存放在桶中,并且除了最后一个桶外,其他的桶都是填满的,因此可以取 $H_r(r)=\lfloor r/b\rfloor+t$,$b\in\mathbb{Z}^+$,$t\in\mathbb{Z}$,其中 b 代表桶的容量. 设 $k=\lfloor r/b\rfloor$,则第 m 个桶的值域为

$$dom_r(B_m)=[k(m-1)+t,\ k(m-1)+t+(b-1)].$$

当选取 v 作为关键字时,由于无法详细预知属性值的分布特性,故需要通过线性 Hash 的方法,选取某个初始 Hash 函数 H_0,通过多次分裂操作最终得到 H_i,这样,我们有下面的定理:

定理 1. 若对任意的 $i\in\mathbb{N}$,H_i 可序列化,则 Hash 函数 H_v 可序列化.

证明. 根据桶指针 n 取不同值分别进行讨论.

当 $n=0$ 时,$H_v(v)=H_i(v)$,显然 H_v 可序列化.

当 $n\neq0$ 时,桶区间 $[0,M)$ 可被分为 3 个部分.

设 m 为桶的编号,根据线性 Hash 的定义,可知:

$$H_v(v)=\begin{cases}H_i(v)=H_{i-1}(v),\ m\in[0,n),\\ H_{i-1}(v),\ m\in[n,2^{i-1}N),\\ H_i(v)=H_{i-1}(v)+2^{i-1}N,\\ \quad m\in[2^{i-1}N,M).\end{cases}$$

显然,这 3 个部分的 Hash 函数均可序列化,并且上式可以归纳为

$$H_v(v)=H_{i-1}(v),\ m\in[0,2^{i-1}N) \tag{7}$$

$$H_v(v)=H_i(v),\ m\in[0,n)\cup[2^{i-1}N,M) \tag{8}$$

因此只需证明 $m\in[n,M)$ 时,H_v 是可序列化的,即对任意的 $p\in[n,2^{i-1}N)$ 和 $q\in[2^{i-1}N,M)$,$dom_v(B_p)\cap dom_v(B_q)=\varnothing$. 由于 q 所在的区间由编号为 $q-2^{i-1}\in[0,n)$ 中的桶分裂而来,设分裂前的桶标记为 B',则 $dom_v(B_q)\subset dom_v(B'_{q-2^{i-1}})$. 又因为式(7)是可序列化的,所以 $dom_v(B_p)\cap dom_v(B'_{q-2^{i-1}})=\varnothing$,因此得证.

综上所述,在整个连续区间$[0,M)$中,Hash 函数 H_v 是可序列化的. 证毕.

定理 1 保证分裂的过程中 Hash 函数的可序列化性.

3 一种基于属性值的桶划分算法

3.1 Hash 函数的设计

为了实现属性值的范围查询,必须选取能够序列化的 Hash 函数,显然使用 Litwin 提出的取模函数生成的桶无法序列化. Robinson[12] 虽然提出了按位取倒序的思想,然而直接使用该函数,无法有效的支持范围查询. Higuchi 等人提出的除法并取反(divide and reverse)方法可支持范围查找[13],然而该方法存在以下的不足:一方面它没有给出明确的初始 Hash 函数以及分裂函数的定义,另一方面它考虑关键字有效位可能出现更新的情况,人为地指定了关键字的总位数,且关键字总位数应远远大于关键字的最大值的有效位,此时将造成无效的分裂操作,即原始桶中的数据经过分裂操作后没在分布到新的主桶中,这样不但增加了 Hash 表的创建时间,而且将产生大量的溢出桶,造成空间的极大浪费.

由于列存储通常应用于"读优先"的海量数据分析型应用环境,一方面存储性能非常重要. 另一方面,属性值一旦确定取值范围就不会频繁地发生更改. 针对上述问题和特点,本文提出了一种可序列化的 Hash 函数,使之不仅能够满足属性值的范围查询效率并且有效提高存储效率. 本文基本思想如下:以最大属性值的有效位作为所有属性值的总位数,并以此设计分裂函数,使分裂后的数据尽可能地分布到新的主桶中,减少无效分裂次数,从而减少溢出桶的个数. 为了方便描述,本文中属性值的数据类型为自然数. 若属性值为其他数据类型,可以通过保序的完美 Hash 函数[14] 转化成自然数. 在按位取倒序[12] 基础上,本文提出改进的 Hash 函数声明如下:

设 $a(i,j)$ 表示对有效位为 i 的自然数 j 按位取倒序运算,并规定 $a(0,0)=0$,对任意的 $i>0$,有:

$$a(i,j)=\begin{cases}a(i-1,j/2),\ j\text{ 为偶数};\\ a(i-1,(j-1)/2)+2^{i-1},\ j\text{ 为奇数}.\end{cases} \tag{9}$$

设 V 是属性值集合,$v_{\max}$ 表示属性值集合中的最大值,则存在正整数 p,使得 $v_{\max}\in[2^{p-1},2^p)$,设 $f(v)=\left\lfloor\frac{v}{2^{p-i}}\right\rfloor$,则定义 H_v 的分裂函数如下:

$$H_i:v\mapsto a(i,f(v)).$$

函数 $f(v)$是一个除法操作,因此它是可序列化的. 正整数 p 是 $v_{\max}$的有效位长度,它保证分裂函数能够使原始桶中的数据在分裂后能够存放到新的主桶中. 此外,分裂函数 H_i 是可序列化的,证明如下:

易知,函数 $f(v)$的值域 $F=[0,2^i)$,函数 $a(i,f(v))$的值域 $A=[0,2^i)$,且 $F\mapsto A$ 为完全映射. 又因为 $f(v)$可序列化,因此 $a(i,f(v))$可序列化,命题成立.

由定理 1 可知,H_v 是可序列化的 Hash 函数,并且序列化函数为 $a(i,H_v)$.

若把桶的编号以二进制的形式表示,则由该分裂函数生成的桶分布如图 2 所示. 其中只有叶子结点才是桶最终的物理编号,非叶子结点是整个分裂过程的中间结果.

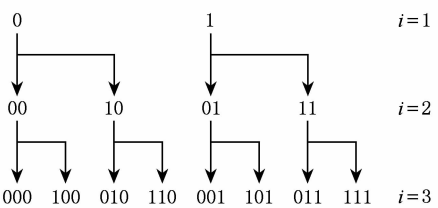

Fig. 2 Bucket partition by split functions.

图 2 由分裂函数生成的桶分布图

若通过函数 $a(i,j)$把每个结点转换成逻辑编号,则图 2 就转换成类似前缀 Hash 树(PHT)[15] 的结构,进一步验证 Hash 函数 H_v 可支持范围查询.

3.2 桶划分算法

根据 3.1 节定义的 Hash 函数,算法需要如表 1 所示的全局变量:

Table 1 Global Variables Defined in the Algorithm

表 1 算法中定义的全局变量

Global Variables	Description
count	Number of data items
b	Capacity of each main bucket
M	Number of current main buckets
b'	Capacity of each overflow bucket
M'	Number of current overflow buckets
limit	Bucket split factor
p	Valid bits of v_{max}
i	Bucket split level
n	Bucket spilit pointer

算法由下面的 4 个函数组成:数据插入函数、数据查找函数、桶分裂函数和 Hash 表创建函数组成. 4 个函数的代码描述如下:

算法 1. 数据插入函数.

Function *HashInsert*(m,*key*).

输入:待插入的桶编号、关键字.

```
① Begin
② If !isFull(bucket[m])
③ insert(key);
④ Else
⑤ If isNull(bucket[m].overflow)
⑥ malloc(bucket[m].overflow);
⑦ M′++;
⑧ End If
⑨ HashInsert(bucket[m].overflow,key);
⑩ End If
⑪ End
```

数据插入函数向指定编号的桶中插入一条关键字,若主桶存满则插入到溢出桶中,并且增加溢出桶的个数.

算法 2. 数据查找函数.

Function *HashSearch*(*key*).

输入:待查找的关键字;

输出:关键字所在桶的编号.

```
① Begin
② m=a(i,key≫(p−i));
③ If m≥M
④ m=a(i−1,key≫(p−i+1));
⑤ End If
⑥ Return m;
⑦ End
```

数据查找函数建立在式(5)的基础上,它的作用是根据关键字查找对应的桶编号.

算法 3. 桶分裂函数.

Function *HashSplit*()

```
① Begin
② If n==0
③ i++;
④ End If
⑤ M++;
⑥ temp=bucket[n];bucket[n]=null;
⑦ For Each key in temp
⑧ HashInsert(HashSearch(key),key);
⑨ End For
⑩ overflow=bucket[n].overflow;
⑪ While(overflow! =null)
⑫ For Each key in overflow
⑬ HashInsert(HashSearch(key),
   key);
⑭ End For
⑮ M′−−;
   overflow=overflow.overflow;
⑯ End While
⑰ End For
⑱ n=(n+1)%2^(i−1);
⑲ End
```

桶分裂函数把分裂指针指向的主桶中的关键字以及对应的溢出桶中的关键字重新划分,调用算法 1 定义的数据插入函数和算法 2 定义的数据查找函数使这些关键字重新分布到原先的主桶和新的主桶中.

算法 4. Hash 表创建函数.

Function *HashCreate*(b,b',max,*limit*).

输入:主桶和溢出桶容量,属性值的最大值,分裂阈值.

```
① Begin
② M,M′,count,i,n=0;
③ p=⌈lb v_max⌉;
④ For Each item in D
⑤ m=HashSearch(item.v);
⑥ HashInsert(bucket[m],item.v);
⑦ count++;
```

⑧ If $count/(bM+b'M')>limit$

⑨ $HashSplit()$;

⑩ End If

⑪ End For

⑫ End

Hash 表的创建函数是整个算法的总控函数.对于属性值集合 D 中的每个元素,先调用算法 2 定义的数据查找函数确定关键字对应的桶编号,然后调用算法 1 定义的数据插入函数把关键字插入到对应的桶中,当式(4)定义的装载因子超过规定的阈值时,调用算法 3 定义的桶分裂函数执行分裂操作.

4 实验分析

4.1 创建性能和范围查询性能比较实验

实验数据来自星型模式基准数据集[16],该数据集中共有 1 张事实表和 4 个维表.本实验从事实表 Lineorder 的主键列 Orderkey 和 4 个外键列 Custkey,Partkey,Suppkey 和 Orderdate 中依次抽取定量的数据,抽取的数据量最小值为 10^5 个数据项,并且数据量按照 10^5 依次递增.对每次抽取的各列分别创建 B^+ 树(BT)、传统的 Hash 表(HT)和可范围查询的 Hash 表(RHT),统计它们的创建效率和范围查询效率,查询的范围为列值 $v\in[10^8,10^9]$ 的所有数据项.每次抽取的最终实验数据来自各列的创建效率和范围查询效率的平均值.

此外,各数据结构的基本设置为:B^+ 树的每个结点能够容纳 400 个数据项;Hash 表的每个主桶能够容纳 400 个数据项,每个溢出桶能够容纳 320 个数据项,其中 HT 的主桶个数为 5 000,RHT 的分裂阈值为 0.75.

由图 3(a)可见 ,随着数据量的增大,B^+ 树的创建时间将大幅度增加,而 Hash 表创建时间的增长幅度远小于 B^+ 树,其中 RHT 的创建时间略大于 HT,额外的开销是由 RHT 创建过程中的桶分裂操作造成的.图 3(b)对 3 种结构的范围查询性能进行了比较,由于 HT 需要对所有的主桶和溢出桶进行扫描,因此查询时间开销最大,而 B^+ 树和 RHT 只需要对部分的结点或桶进行扫描,因此查询开销大大降低.

实验结果显示,RHT 综合了 HT 的创建性能和 B^+ 树的范围查询性能上的优点,为属性值副本的快速创建和有效地范围查询提供了保障.

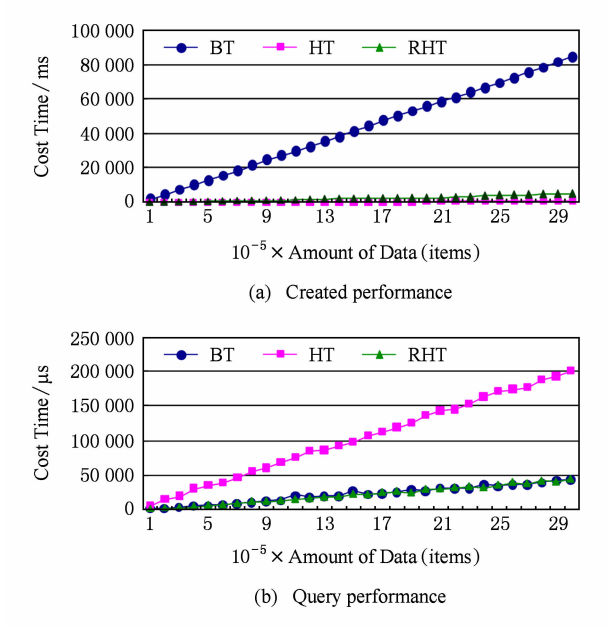

Fig. 3 Comparison of range query performance among B^+-tree, HT and RHT.

图 3 B^+ 树、HT 和 RHT 创建和查询性能比较

4.2 存储性能比较实验

在 4.1 节实验的基础上,从 Lineorder 表的 Custkey 列中随机抽取 300 万条数据.对这些数据分别建立 RHT 和 Higuchi 提出的保序线性 Hash 表(OPLHT)[13],其中 Hash 表的每个主桶能够容纳 400 个数据项,每个溢出桶能够容纳 320 个数据项,分裂阈值为 0.75,属性值的总位数为 32,最大属性值的有效位为 16.

Table 2 The Comparison Between RHT and OPLHT(300 000 integers)

表 2 RHT 和 OPLHT 的比较(300 000 数据量)

Type of Hash Table	Main Buckets	Overflow Buckets	Cost of Creation
OPLHT	2 670	10 244	247 906
RHT	9 800	1 255	5 616

表 2 显示了在 RHT 和 OPLHT 各项指标的比较,包括主桶个数、溢出桶个数和创建开销(ms).虽然使用 OPLHT 创建的 Hash 表主桶个数远小于 RHT,但是溢出桶个数远大于 RHT,并且主桶和溢出桶的总空间大于 RHT,造成存储空间的极大浪费,这是由于无效的分裂操作产生的.同时,无效的分裂操作和大量的溢出桶导致了 Hash 表创建性能的低下,使得 OPLHT 的创建开销约为 RHT 的 200 倍.

实验结果显示,RHT 无论是存储效率还是创建性能上都优于 OPLHT,为属性值副本的有效存储提供了保障,同时进一步确立了它高效的创建性能.

5 相关工作

20 世纪 80 年代初,线性 Hash 的首次提出取得了两方面的成就:首先,有效地减少了溢出桶的个数;其次,为 Hash 表提供了一种新的桶划分策略. Ramomohanarao 提出了一种可递归的线性 Hash[17],用于划分溢出桶的等级. Robinson 提出了一种保序的线性 Hash[12],使用除法运算和位倒序的方法构造分裂函数,使得 Hash 表中的桶具有一定的顺序. Kriegel 在此基础上进行了扩展,提出了一种多维的线性 Hash 函数[18],使其能够有效地控制文件的增长,由于这些 Hash 函数的值域为 0～1 之间的实数,它们无法处理任意类型的数据.

20 世纪 80 年代中期,由于 Hash 表只能进行精确查询的局限性,学术界进行了 Hash 表的范围查询的研究. Chang 提出了一种基于桶地址 Hash 的正交范围查询算法[19],通过该算法可以计算出关键字被划分到不同的桶地址空间的概率. 此外,他结合线性 Hash 的方法对正交范围查询算法进行了改进[20],提高了查询性能. 但是这两种查询都是基于概率的,存在不确定因素,因此它们无法保证查询结果的精确性.

21 世纪初,随着网络技术的发展,Hash 表被广泛地用于点对点网络中,Ratnasamy 等人提出分布式 Hash 表(DHT)的思想[21],并在未来的 10 年内被广泛应用与研究. Ramabhadran 等人在 DHT 的基础上提出了一种前缀 Hash 树(PHT)的结构[15],使 DHT 能够支持范围查询. Higuchi 结合线性 Hash 的特点,再次使用保序线性 Hash 的方法构造分裂函数[13],使 Hash 表中的关键字能够根据前缀划分到各个桶中,从而实现了精确的范围查询,但是如 3.1 节所述,该方法会造成极大的空间浪费. 张旺光等人提出了 M+树的结构[22],把 B^+ 树索引、Hash 表和二叉排序树结合起来,实现关键字的范围查询. 使用这种方法最大的缺点是需要建立二叉排序树作为虚拟映射树,并且结点的个数与关键字的个数一致,不但造成存储空间的浪费,而且影响了索引的创建性能.

6 结束语

本文提出了一种适用于范围查询的列存储数据桶划分模型,并且在改进保序的线性 Hash 方法的基础上,设计了一种基于属性值桶划分算法. 使用该算法能够快速地构建可序列化的 Hash 表,减少需要遍历的桶数,提高了列存储数据中属性值副本的创建性能、存储性能和范围查找的性能. 但是,使用这种方法构建的 Hash 表仍然存在以下的不足:首先,无法彻底避免溢出桶;其次,当属性值分布极不均匀时,或者存在大量的重复值时,某些主桶可能拥有数量较多的溢出桶,某些主桶可能会成为空桶,造成空间的大量浪费. 我们将在以后研究这些问题,并且制定相应的解决方案.

参 考 文 献

[1] Abadi D J, Boncz P A, Harizopoulos S. Column oriented database systems [C] //Proc of the 35th VLDB Conf. Trondheim, Norway: VLDB Endowment, 2009: 1664-1665

[2] Zukowski M, Boncz P A, Nes N, et al. Monet DB/X100——A DBMS in the CPU cache [J]. IEEE Data Engineering Bulletin, 2005, 28(2): 17-22

[3] Stonebraker M, Abadi D J, Batkin A, et al. C-Store: A column-oriented DBMS [C] //Proc of the 31st VLDB Conf. Trondheim, Norway: VLDB Endowment, 2005: 553-564

[4] MacNicol R, French B. Sybase IQ Multiplex-Designed For Analytics [C] //Proc of the 30th VLDB Conf. Trondheim, Norway: VLDB Endowment, 2004: 1227-1230

[5] Abadi D J, Madden S, Hachem N. Column-stores vs. row-stores: How different are they really? [C] //Proc of the 2008 ACM SIGMOD Int Conf. New York: ACM, 2008: 967-980

[6] Garcia-Molina H, Ullman J D, Widom J. Database System Implementation [M]. Upper Saddle River, NJ, Prentice-Hall, 2000

[7] Chan C Y, Ioannidis Y E. Bitmap index design and evaluation [C] //Proc of the 2008 ACM SIGMOD Int Conf. New York: ACM, 1998: 355-366

[8] Olson M A, Bostic K, Seltzer M I. Berkeley DB [C] //Proc of the 10th USENIX Annual Technical Conf. Berkeley, CA: USENIX, 1999: 183-191

[9] Maurer W D, Lewis T G. Hash table methods [J]. ACM Computing Surveys(CSUR), 1975, 7(1): 5-19

[10] Litwin W. Linear hashing: A new tool for file and table addressing [C] //Proc of the 6th VLDB Conf. Los Alamitos, CA: IEEE Computer Society, 1980: 212-223

[11] Litwin W. Linear hashing: A new algorithm for files and tables addressing [C] //Proc of the 1st ICOD Conf. London: Heyden & Son, 1980: 260-276

[12] Robinson J T. Order preserving linear hashing using dynamic key statistics [C] //Proc of the 5th ACM SIGACT-SIGMOD Symp on Principles of Database Systems. New York: ACM, 1986: 91-99

[13] Higuchi K, Tsuji T. A linear hashing enabling efficient retrieval for range queries [C] //IEEE Int Conf on System, Man and Cybernetics 2009. Piscataway, NJ: IEEE, 2009: 4557-4562

[14] Fox E A, Chen Q F, Daoud A M. Order preserving minimal perfect Hash functions and information retrieval [J]. ACM Trans on Information Systems(TOIS), 1990, 7(1): 279-311

[15] Ramabhadran S, Ratnasamy S, Hellerstein J M, et al. Brief announcement: prefix Hash tree [C] //Proc of the 23rd ACM Symp on Principles of Distributed Computing. New York: ACM, 2004: 368

[16] Abadi D J, Myers D S, DeWitt D J, et al. Materialization strategies in a column-oriented DBMS [C] //Proc of ICDE 2007. Piscataway, NJ: IEEE, 2007: 466-475

[17] Ramamohanarao K, Davis R S. Recursive linear hashing [J]. ACM Trans on Database Systems (TODS), 1984, 9(3): 369-391

[18] Kriegel H P, Seeger B. Multidimensional order preserving linear hashing with partial expansions [C] //Proc of the 1st Int Conf on Database Theory. Berlin: Springer, 1986: 203-220

[19] Chin-Chen Chang, Chen C Y. Orthogonal range retrieval using bucket address hashing [C] //Proc of the 4th Int Working Conf on Statistical and Scientific Database Management. Berlin: Springer, 1989: 133-140

[20] Chen C Y, Chin-Chen Chang, Lee R C T, et al. Optimal linear hashing files for orthogonal range retrieval [C] //Proc of the 20th Computer Software and Applications Conf. Los Alamitos, CA: IEEE Computer Society, 1996: 406-413

[21] Ratnasamy S, Francis P, et al. A scalable content-addressable network [C] //Proc of the 2001 Conf on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM, 2001: 161-172

[22] Zhang Wangguang, Zhuang Yi. M+-tree: A novel and effective dynamic Hash algorithm [J]. Journal of Computer Engineering, 2004, 30(16): 94-96 (in Chinese)
(张旺光, 庄毅. M+-树:一种新型、高效的动态 Hash 算法 [J]. 计算机工程, 2004, 30(16): 94-96)

Li Yefeng, born in 1984. PhD candidate. His main research interests include database, distribution technology, and information security.

Le Jiajin, born in 1951. Professor and PhD supervisor. Member of China Computer Federation. His main research interests include database, data warehouse, software engineering theory and practice.

Wang Mei, born in 1980. PhD. Her main research interests include database, image semantic analysis, and information retrieval.