1990年1月 重庆大学学报 Vol.13,№.1
第13卷第1期 JOURNAL OF CHONGQING UNIVERSITY Jan.1990

数据压缩的hash方法

DATA COMPRESSION WITH HASH CODES

谭强明

Tan Qiangmin

（计算机系）

摘 要 在数据库应用中常要求数据压缩存贮。所采用的压缩方法应当有较小的系统开销和期望的压缩效果。文章给出了一种用其hash函数值作为编码来取代数据本身以减少文件的数据存贮量的压缩方法。该方法对许多应用都有较好的压缩效果，并其编码/解码的成本以为对hash表一次查找或直接访问的开销。文章也讨论了hash编码数据压缩方法的实现技术，以及编码和解码的算法。

主题词 数据库；数据压缩；数据存储；文件系统；算法；编码/文件结构
中国图书资料分类法分类号TP3P2

ABSTRACS In many database applications, data have to be compressed before they can be stored. Data compression techniques with less overhead and better space saving are expected. This paper gives a data compression technique by which the size of a file may be reduced through substituting codes for its field values. The codes are the corresponding locations of the values in the hash table that is comprised of all the values of the field. The compression is effective for many applications, and the cost of the coding/decoding is equal to that of one access to the hash table. The implementation of the technique as a compression tool in DBMS for data storage and the algorithms of the coding/decoding are discussed.

SUBJECT WORDS data libraries; data compression; data storage; file systems; algorithm; coding/file

一、引 言

许多微机数据系统，其数据的存贮量往往超过计算机有限的存贮空间（硬盘）。解决这一问题可有下列的途径：

合理的数据库设计，减少数据冗余。

收文日期 1988年11月25日

· 把不常用的数据脱机，例如保存在磁带上，在要求使用时才装入。

· 增加新的硬盘。

· 要用数据压缩技术。

考虑使用何种途径来解决存贮空间不足的问题应视具体的对象而定。但对数据进行压缩存贮无疑是优先选择的，既经济又有效的一种途径。

数据压缩的方法多种多样，但基本上都是利用数据的重复特征进行压缩[1-7]。压缩的效果与特定的应用有关，并且对系统效率的影响也各有不同。一般地，无论采用何种压缩技术，系统效率的降低是不可避免的，设T_0是系统允许的最大响应时间，σ_0是期望的数据压缩（允许数据存贮的空间与数据存贮量之比），那么，所选用的数据压缩技术应满足：其数据压缩率$\sigma \leqslant \sigma_0$，并且压缩后系统的响应时间$T \leqslant T_0$。如果数据压缩不能满足这一要求，就只有另辟途径。

下列的情况是数据库应用常见的。设R是一数据库文件（关系），其记录数为N，有这样的域（属性）满足域长$l > 3$，域值个数$M < N$。显然，该域的值在R中有多次重复，可以对它们进行某种编码（编码长$< l$），然后用这些编码替换对应的域值，以获得对R的压缩存贮。对R的编码存贮应对用户透明，用户在使用R时不应感到有任何编码的存在。因此，除了对存贮数据进行编码外，在输出时还应做解码，即把编码转换成原有的数据形式。无论编码和解码都会产生系统额外的开销，增加系统的响应时间。对文件压缩存贮所获得的好处能否抵偿由此而引起的系统效率的下降，压缩后系统的响应时间是否超过允许的限度——这成为系统设计者决定对数据进行压缩存贮时先要考虑的问题。以前许多数据压缩方法都是利用数据本身的重复性进行压缩，而很少考虑许多应用中数据之间存在着大量重复这一事实[2,4,5,6]。本文所提出的方法对文件中数据大量重复的域进行压缩，并采用 hash 表技术进行编码和解码，是一种有效简单而应用面较广的数据压缩方法。Hash编码压缩的基本思想是把文件中要压缩的属性值放入一hash表中，用它们在该表中的存放位置作为对应的编码。去取代文件中相应数据的存在。作者在文章中讨论了该方法在数据库系统中的实现技术以及编码和解码的算法，并且分析了它的压缩效果及其对系统响应时间的影响。

二、压缩方法

设有一有限集合Z，$||Z|| = M$，h是Z到整数集合$[1:M]$上的hash函数，$H[1:M]$是一存贮空间，将每个$x \in Z$按下列方法存贮后x所在H中的位置称为x的hash编码：

① 计算$n = h(x)$，$n \in [1:M]$

② 若$H(n)$为空，则$H(n) \Leftarrow x$；否则在H中任找一空单元其位置为m，$H(m) \Leftarrow x$。

我们把已存入了Z的存贮空间H称为在Z上的hash表。显然，在H中，每个x都有唯一确定的位置，而每个位置也有一对应的x。亦即存在一个由hash表H确定的由Z到$[1:M]$的置换$\pi = \begin{pmatrix} x_1 & x_2 \cdots\cdots x_M \\ n_1 & n_2 \cdots\cdots n_M \end{pmatrix}$，这里$n_i$是$x_i$在$H$中的位置或曰hash编码。我们记$n = \pi(x)$为$x$在$H$上编码，而$x = \overline{\pi}(n) = H(n)$为$x$在$H$上解码。

一般的，π对Z是不唯一的，随着生成 hash 表时取x的顺序不同而变化，并且$h(x) \neq$

$\pi(x)$。但若$h(x)$在$[1:M]$上均匀分布，即$h(x)$为Z到$[1:M]$的一一对应函数，则有唯一的π，并且$\pi(x)=h(x)$。

设数据文件R的域f_i的域值空间为Z_i，H_i是Z_i依赖于hash函数h的hash表。π_i为由H_i确定的Z_i到$[1:M]$的置换，将f_i的每个值x替换成$\pi_i(x)$的过程，称为对R在f_i上的hash编码压缩。压缩后的数据文件R'记为$\pi_i(R)$，压缩后的域f_i'记为$\pi_i(f_i)$，其域值空间为$[1:M]$。对R多个域进行hash编码压缩是可能的，同样有

$$R'=\pi(R)=\pi_{i1}(\pi_{i2}(\pi_{i3}\cdots\pi_{ik}(R)\cdots))$$

称为对R的hash编码压缩。

三、压 缩 率

讨论用hash编码对数据压缩存贮的效果，我们有如下的定义：

期望压缩率σ_0，$\sigma_0=K/S_R$

这里S_R是文件R的数据量(byte)，K是期望R存贮的空间大小。在要求数据压缩的场合，有$0<\sigma_0<1$，$1\ll S_R$。

实际压缩率σ，$\sigma=S_{\pi(R)}/S_R$

这里$S_{\pi(R)}$是对R压缩后数据的存贮量，它等于$\pi(R)$的大小与所增加的hash表大小之和。σ与σ_0之间关系为$\sigma<\sigma_0$。

对域f_i的压缩率σ_i，$\sigma_i=S_{\pi_i(R)}/S_R$

这里$S_{\pi_i(R)}$是对在f_i上压缩后的数据存贮量。在f_i上进行有效的数据压缩要求$\sigma_i<1$。

设f_i的域长为l_i，$\overline{l_i}$为f_i值的平均长度，P为编码长度，N为R记录的总数，M_i为f_i不同域值的个数，L为每个记录的长度，显然

$$\sigma_i=\frac{N(L-l_i+P)+M_i\overline{l_i}}{NL}=1-\frac{l_i-P}{L}+\frac{M_i\overline{l_i}}{NL}$$

要使$\sigma_i<1$，则有下列不等式成立：

$$\frac{M_i}{N}<\frac{l_i-P}{\overline{l_i}}$$

一般有 $l_i=\overline{l_i}$，使得$(l_i-P)/\overline{l_i}<1$，因此$M_i<N$就有$\sigma_i<1$。若$l_i>\overline{l_i}$，既使$M_i=N$时，也有$\sigma_i<1$。

对于σ_i，当$N\ll M_i$并且$L>l_i$时，σ_i近似有

$$\sigma_i=1-\frac{l_i-P}{N}$$

因此，在$N\ll M_i$和$L>l_i$时，σ_i仅与l_i相关（假定L不变），即l_i越大，压缩效果越好。

下面分析σ与σ_i的关系，设对R的k个域进行了hash编码压缩，因此有

$$\sigma=\frac{N\left(L-\sum_{i=1}^{k}(l_i-P)\right)+\sum_{i=1}^{k}M_i\overline{l_i}}{NL}$$

$$= \frac{\sum_{i=1}^{k} N(L-l_i+P) + \sum_{i=1}^{k} M_i\overline{l_i} - (k-1)NL}{NL}$$

$$= \sum_{i=1}^{k} \frac{N(L-l_i+P)+M_i\overline{l_i}}{NL} - k+1$$

$$= \sum_{i=1}^{k} \sigma_i - k+1$$

要求$\sigma<\sigma_0$，就有

$$\sum_{i=1}^{k} \sigma_i < \sigma_0 + k - 1$$

四、压缩的实现

(一)Hash表存贮结构

在使用了hash编码压缩技术的系统中，hash表作为相联数据文件的一部分，也应作为一磁盘文件来存贮。由于操作系统访盘时间远大于内存数据的存取时间，因此，评价文件的访问效率将以访盘的次数为标准。考虑到操作系统读写文件的特点，hash表的文件结构如图1所示。文件分为G个组，每个组又由E个块组成（块的尺寸为操作系统一个磁盘读写块的尺寸）。Hash函数把每个压缩数据x映射到唯一的组：在每个组内，x顺序存放。当一个组存满之后，文件在自由存贮区获取一溢出组，并把放入这个溢出组内。文件中每个组最后两字节$g-ptr$作为指向溢出组的指针。

Hash表可具有定长记录和变长记录两种格式。变长记录除了存放x本身外，前面还加一记录长度域。变长记录hash表对于压缩属性值长度变化范围大的域十分有效。

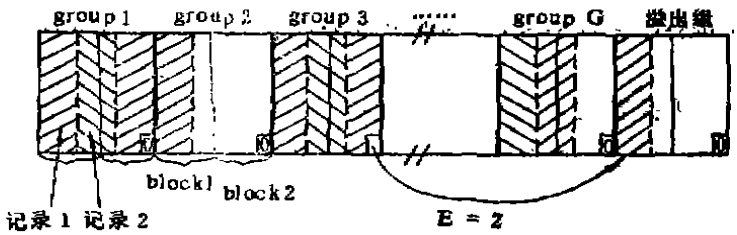

图1 Hash表文件结构

(二)编码解码算法

如此实现的hash表文件结构决定了x的hash编码是一二元组(g,e)，其中g是x所在的组，e是其为该组的第n几个记录。对x编码即找x在hash表的位置(g, e)，其算法如下：

CODING(x, g, e)

（1）$g=h(x)$，$e=1$。

（2）置读指针到第g组。

（3）读一记录y，如果有$y=x$，返回(g, e)退出。

（4）如果为空，若hash表为变长记录格式，计算x的长度，形成x的变长记录写入；否则以x为一定长记录写入。返回(g, e)退出。

（5）若当前组读完，如果溢出组指针$g-ptr$为空，由自由存贮区取一溢出组，置g和$g-ptr$为溢出组的组号；否则，即$g-ptr$存在，置$g=g-ptr$，$e=0$，并置读指针到第g组。

（6）$e=e+1$，转（3）。

对x的解码即由其hash编码(g, e)查找x，其算法为：

DECOING(x, g, e)

（1）如果为定长记录则直接置读指针到第g组的第e个记录处；否则

（2）若$e=1$，重复做：读记录长度，由此置读指针到下一记录并且$e=e-1$。

（3）读记录x。

删除hash表中的记录将使同一组或相连溢出组在该记录后面的所有记录向前移动。下面是按编码删除x的算法：

DELETE(g, e)

（1）执行DECODING(x, g, e)获得x和x记录开始位置B。

（2）重复做：清空B开始的记录。置读指针$E=B+$当前记录长度并读E开始的记录为新的当前记录。若当前记录为空则退出，否则写到B，置$B=E$。任何时候若组结束出现，置读指针到$g-ptr$指向的组。

删除x将使hash表中x以后记录位置均产生改变，亦即它们的hash编码发生改变。为了保证数据的正确性，必须对与hash表相联的数据文件中对应的hash编码做相应的修改。

由于对数据文件修改可能将耗费大量的开销，在删除hash表中的记录时可以考虑不做移动，而只在被删除的记录上做一删除标志。这样，只需在一定的时间后重组hash表，便可减少大量的对相联数据文件修改的开销。

设hash表中每个组的平均记录数为β，那么由前面给出的算法可知，对于编码读写记录的平均次数为$\frac{1}{2}(\beta+1)$；解码定长记录为1，变长记录为$\frac{1}{2}(\beta+1)$。删除将有β次读记录和平均$\frac{1}{2}(\beta+1)$次写（没有考虑对相联数据文件的改动）。如果删除不做记录移动，则该操作的平均读写记录的次数只为解码的读写次数加1。对hash表中某个记录的修改由删除被修改记录DELETE(g, e)和编码修改值CODING(x, g, e)组成。其平均读写记录的次数等于删除和编码读写次数之和。

（三）实现方法

在数据库管理系统中实现hash编码压缩技术很简单，在操作系统和数据库管理系统之间加一编码解码介面即可，如图2所示。该介面的功能是完成对由数据库读出数据的解码，以及写入数据库前对数据的编码。

编码和解码操作本身是透明的，并不引起任何用户应用程序的改变。因为施加在压缩文

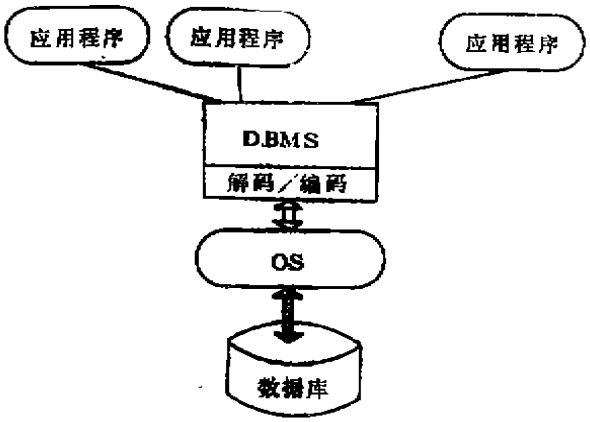

图2 Hash编码压缩在数据库管理系统中实现

件$\pi(R)$上的任何操作在解码/编码后，完全与施加在未压缩文件R上相同。Hash编码压缩技术在数据库管理系统中实现，将使该管理系统增加数据压缩的功能。下列的命令提供用户压缩数据，以及维护hash表使用。

c_r-hf〈文件〉，〈属性名〉，〈hash表名〉，G，E，〈格式〉，〈选择〉，〈装入内存〉{，〈自编hash函数入口〉}

对文件的指定属性进行数据压缩。这里〈hash表名〉是生成hash表的文件名。G是hash表的组划分数，E是每组块数。〈格式〉说明hash表是定长或是变长记录。〈选择〉给出是否打印输出hash表中记录的分布、空间的利用率等信息。〈装入内存〉规定是否在读写联数据文件时把整个hash表放入内存。〈自编hash函数入口〉可缺省，缺省由系统定义的hash函数计算。

$ins-hf$〈hash表名〉，〈值串〉

在hash表中增加数据。

$vpd-hf$〈hash表名〉，〈值〉，〈修改值〉

把hash表中的〈值〉修改成〈修改值〉。

$clr-hf$〈文件名〉，〈属性名〉，〈选择〉

对压缩属性进行清理。去掉相联hash表中无用的数据。〈选择〉说明是否打印输出hash表中记录的分布、空间的利用率等信息。

$res\rightarrow hf$〈文件名〉，〈属性名〉

恢复对属性的压缩，去掉与〈属性名〉相联的hash表文件。

五、对系统效率的影响

设在数据文件R上的k个域进行了hash编码压缩。压缩后的数据文件$\pi(R)$每个记录的平均访盘次数。

$$u=\frac{L-\sum_{i=1}^{k}(l_i-P)}{B}+\sum_{i=1}^{k}w_i \qquad (1)$$

这里$\left(L-\sum_{i=1}^{k}(l_i-P)\right)/B$为访问$\pi(R)$本身每个记录的平均访盘次数，$w_i$为对$f_i$值编码解码的平均访盘次数，$B$为一个磁盘读写块的大小。由于对未压缩文件的每个记录平均访盘次数为L/B，故压缩后所增加的系统开销为：

$$\Delta u = \sum_{i=1}^{k} (w_i - (l_i - P)/B) \qquad (2)$$

由上式可知，要减少Δu，须减少w_i，即提高对hash表记录的平均访问速度，或减少对hash表记录的平均读盘次数。要做到这一点，除了正确地选择$h(x)$使f_i上的域值在hash表中均匀分布之外，还须在建立hash表时，适当地选择文件的组数G和每个组的块数E，使得既不浪费存贮空间，又使每个组的记录所占的平均块数尽量少。因此，设β_i为与f_i相联的hash表每个组的平均记录数

$$\beta_i = M_i/G_i$$

在选择G_i和E_i时须要遵循下列建议：

· 使$G_i \leqslant M_i$，即$\beta_i \geqslant 1$，因为每个记录只能装在一个组内，$G_i > M_i$则浪费空间。

· E_i尽可能为 1，并且使每组记录的平均长度$\beta_i\overline{l_i}$尽量为B的整倍数E_i。这里$\overline{l_i}$在定长记录hash表中为f_i的域长，在变长记录hash表中为f_i值的平均长度，$\overline{l_i} \leqslant l_i$。

· G_i最好为质数，并且不为 2 和 5 的倍数。因为大多情况下hash函数均以G_i作除取余。

在hash表的每个组中，记录所占的平均块数为$\beta_i \cdot \overline{l_i}/B$。记$n_i = \lceil \overline{l_i}/B \rceil$，$\lceil\ \rceil$表示取不小于$\overline{l_i}$（平均每个 hash 表记录所占的磁盘块数）的最小整数。由于编码在组内顺序查找，故编码时平均访盘次数w_i为

$$w_i = \frac{\beta_i\overline{l_i}/B + n_i}{2} \qquad (3)$$

解码若定长格式为一次直接访问，故其w_i为

$$w_i = \begin{cases} \overline{l_i}/B & \overline{l_i} \geqslant B \\ 1 - \dfrac{1}{G_i} & \overline{l_i} < B \end{cases} \qquad (4)$$

若变长格式解码为组内顺序查找，故其w_i为

$$w_i = \frac{\beta_i\overline{l_i}/B + n_i}{2} \qquad (5)$$

采用定长记录hash表对文件进行hash编码压缩一般有$\Delta u > 0$（例外的情况是对G_i很小的定长记录hash表解码）。因为一般有w_i最小为1，要使$\Delta u \leqslant 0$，至少要使$1 \leqslant (l_i - P)/B$，即$l_i \geqslant B + P$。由此有$l_i > B$，并且$n_i = 2(l_i = \overline{l_i})$。另一方面，$w_i = 1$，只有$(\beta_i\overline{l_i}/B + n_i)/2 = 1$。由此可得出$\beta_i + 2 < 2$，这是不可能的。同理可推断在$w_i > 1$时均有$\Delta u > 0$。

用变长记录hash表压缩文件有可能使$\Delta u \leqslant 0$。例如，当$n_i = 1$时，$\overline{l_i} \leqslant (2(l - P) - B)/\beta_i$就可使$\Delta u \leqslant 0$。在这种情况下，$f_i$的域长$l_i$将远大于$f_i$值的平均长度$\overline{l_i}$。例如，若$l_i = 300$，$P = 3$，$B = 512$，$\beta_i = 2$，则$\overline{l_i} \leqslant 41$。

另一种减少Δu开销的方法是在hash表较小并且内存有足够的空时把整个hash表一次装入内存。在频繁访问数据文件时这种方法可大大降低访问成本。

六、结 论

Hash编码数据压缩是采用文件的域值在hash表的位置对文件进行编码压缩来减少存贮量的一种方法。压缩与数据值本身的特征无关，因此这种方法可以为数据库系统的一种通用的压缩数据存贮的工具。Hash编码压缩在文件域长l_i较大，并且文件的总记录数N大于域值个数M时，或者域长l_i大于其值的平均长度$\overline{l_i}$时压缩效果较好；并且所增加的系统开销也较小；其一次编码或解码的成本为一次hash查找或一次直接访问的开销。

实际实现的hash表按组划分，每个组由若干块组成，组的划分G和每组块数E的选择直接影响压缩效果和系统效率。选择G和E必须遵循一定的原则使得既获得满意的压缩效果又具有较低的解码编码时间。用户也可以通过列出hash表中记录分布以及空间利用率等信息，随时修改G和E，或把hash表装入内存来改进效率。

除在数据库系统中使用该方法来压缩数据的存贮外，在一般的数据处理应用系统中也可使用该方法。

参 考 文 献

〔1〕 Hoffman, D.A., A method for Construction of Minimum Redundancy Code, Proc. IRE, 1952

〔2〕 Wagel, R.A., Common Phrases and Minimum Space Text Storage, CACM, 1973, 16(3): 148－152

〔3〕 Rubin, F., Experiment in Text File Compression, CACM, 1976, 19(11): 617－623

〔4〕 Kang, K.N.C, et al., Storage Reduction Through Minimal Spanning Trees and Spanning Forests, IEEE Trans. on Comp., 1977, C－26(5): 425－434

〔5〕 Even, S. and Rodeh, M., Economical Encoding of Commas between Strings, CACM, 1978, 21(4): 315－317

〔6〕 Yahiko Kambayashi, et al., Data Compression Procedures Utilizing the Similarity of Data, Proc. ANCC, 1981, 555－562

〔7〕 Davis, R.H., et al., Data Comprssion in Limited Capacity Microcomputer System, Information Processing Letters, 1982, 13(4, 5), 138－141