基于谓词索引的海量数据压缩存储及数据操作算法*)

赵 锴 李建中 骆吉洲
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 压缩数据库技术是海量数据管理的重要技术之一。利用海量数据自身及其数据操作的特点,提出了一种海量数据压缩存储结构。该存储结构将第二级和第三级存储器结合起来,以数据操作条件中的谓词为索引,在减少存储空间的同时有效地支持查询、删除和更新等数据操作。理论分析和实验结果表明,这种存储结构可以提高海量数据的存储效率和数据操作的性能。

关键词 海量数据压缩,谓词索引,存储结构,数据库操作

Storage and Operation of Massive Data Compression Based on Predication Index

ZHAO Kai LI Jian-Zhong LUO Ji-Zhou
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract The compression technique of database plays an important role in the management of massive data. The characteristics of massive data and the data operations on these data are examined, and compression storage architecture of massive data is proposed in this paper. Our architecture utilizes both the secondary and the tertiary storage, and the predications in the condition of operations are used as index. The theoretical analysis and experiment results show that our architecture can reduce the storage space needed remarkably and improve the efficiency of data operation such as data query, data insertion and data deletion.

Keywords Data compression, Predication index, Storage architecture, Data operation

1 引言

随着科技的发展和信息化程度的提高,超过10^{13}字节的海量数据随处可见并迅速增长。海量数据的有效管理和查询技术的研究成为海量数据处理的关键。针对各种海量数据的特点,设计相应的压缩存储和查询算法,以提高其存储和查询处理的效率是当前压缩数据库技术研究的热点。海量数据的压缩存储包括在线压缩存储[2,3,6]和离线压缩存储[1]两部分,广泛应用于数据仓库[7~9,12]和数据挖掘[10]中。

在实际应用中,当在线数据量积累到一定程度后,就将其离线压缩存储于第三级存储器(例如磁带库)。当前,工业界使用的海量数据压缩存储方式[1]多集中在利用文件级的压缩算法(例如 Huffman、Lzw 和算术编码等[5]),将海量数据分块依次压缩存储至第三级存储器。查询时需依次将压缩块从第三级存储器导入第二级存储器,然后解压查询。采用这种压缩方法的前提是海量数据作为备份压缩存储,并不存在删除和更新操作。这种方式只考虑了对数据的压缩存储,缺乏对查询操作的有效支持。

我们在文[1]中首次提出了一种第二级和第三级存储器相结合的海量离线数据压缩存储结构。该结构提高了压缩和查询的效率,但也存在一定的局限性,例如不支持删除和更新操作。

当第二级存储器和第三级存储器紧密结合形成一个存储体后,建立在该存储体上的压缩数据库系统将同时具有第二级存储器的速度和第三级存储器的容量。系统在第三级存储器中压缩存储所有数据的备份。对于当前访问频繁的"热点数据",需从第三级存储器导入,并解压存储在第二级存储器,以提高对这些"热点数据"的在线操作响应时间。"热点数据"作为在线数据,就存在删除和更新操作。因此,系统在提高查询效率和降低存储空间的同时,还必须有效地支持"热点数据"的删除和更新,保证在删除和更新操作后第二级和第三级存储器中数据的一致性。以上正是本文所关注和解决的问题。

本文提出的压缩存储结构,①以数据操作条件中的谓词为索引,可以快速定位满足操作条件的压缩数据在第三级存储器中的位置,从而提高了数据操作的响应时间;②支持删除和更新等操作;③将二级存储器作为缓存,减少了三级存储器到二级存储器的数据传输量;④根据三级存储器磁带库的传输特点,各种数据操作只需一遍顺序扫描完成,避免了多遍反复扫描。

本文其余部分如下组织:第 2 节介绍基于谓词索引的海量数据压缩存储结构及其压缩存储算法;第 3 节是压缩存储结构上的查询、删除和更新等数据操作算法以及操作效率的理论分析;第 4 节是实验结果及其分析;最后是结论。

2 基于谓词索引的海量数据压缩存储结构及其压缩存储算法

该压缩存储结构由第二级存储器和第三级存储器中的存储结构共同组成。

2.1 基本思想

*)本课题得到国家自然科学基金(项目号:60273082)资助。赵 锴 硕士研究生,主要研究方向为压缩数据技术;李建中 博士生导师,主要研究方向为数据库及相关技术;骆吉洲 博士研究生,主要研究方向为压缩数据库技术。

由于对海量数据的操作(查询、更新和删除)需要较高的时间和空间代价,因此有查询类型少、查询语句简单固定、更新和删除频率低等特点。我们希望改善那些使用频繁的数据操作的响应时间。假设对于一个海量数据库应用,经过统计后可以获得频繁使用的数据操作语句,将其 where 子句(即数据操作条件)中出现的所有原子谓词找出来,按照文[1]把它们分为不确定性谓词和确定性谓词两类。

定义 1 不确定性谓词(Indefinite Predication,IP) 数据操作下发前不能完全确定的谓词,比如"用户名= * * * "。这类原子谓词左端的属性名常固定,但右端的属性值可以变化。

定义 2 确定性谓词(Definite PredicationDP) 数据操作下发之前已经确定的谓词,比如"通话时间>5min 的通话记录"。这类原子谓词的属性名和属性值固定,常常作为一个整体出现在操作条件中。

在一般应用中,主键常作为不确定谓词的属性,非主键常作为确定性谓词的属性。由于确定性谓词之间可能存在冗余,给出以下定义:

定义 3 最小冗余的确定性谓词集合(MRDP) 设数据操作(例如查询操作)$Q_1,\cdots,Q_k$ 涉及到的确定性谓词的集合为 $S=\{pred_1,\cdots,pred_m\}$。对于确定性谓词 $pred_i$ 和 $pred_j$,如果任意满足谓词 $pred_i$ 的记录同时也满足谓词 $pred_j$,记为 $pred_i \leqslant pred_j$,我们就将 $pred_i$ 从 S 中删除。最后得到的谓词集合称为最小冗余的确定性谓词集合,仍记为 $S=\{pred_1,\cdots,pred_n\}$。

例如,任意满足确定性谓词"通话时间>10min"的记录同时也满足确定性谓词"通话时间>5min",故保留谓词"通话时间>5min"即可。因此,一个查询条件 C_1 可以用不确定性谓词和具有最小冗余的确定性谓词的组合 C_2 来代替。由 C_2 获得的查询结果是 C_1 的超集,记为 $C_1 \subseteq C_2$。

虽然这些不确定性谓词和最小冗余的确定性谓词是通过统计频繁使用的数据操作语句中得来的、个数也是有限的,但它们的相互组合却可以表示更多的数据操作语句。

经过统计获得所有不确定性谓词和最小冗余的确定性谓词后,在压缩存储时将每一条元组所满足的确定性谓词的属性值,最小冗余的确定性谓词以及删除字段作为索引存储于二级存储器的数据库中。同时将元组存放到相应的待压缩磁盘缓存文件中。当缓存装满后,将其顺序压缩存储至第三级存储器,并在数据库中存储谓词索引与符合该谓词索引的元组所在压缩文件地址的对应关系。

2.2 第二级存储器中的存储结构

二级存储器的存储结构由数据库和磁盘缓存组成。

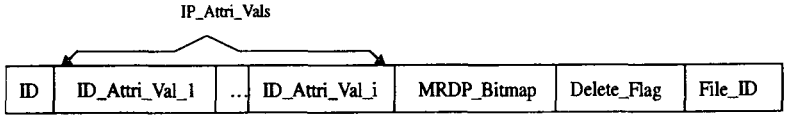
数据库中保存三个表:TempTable 缓存每个压缩文件解压后的元组;Pred_Index 和 Address_Map 相组合作为谓词索引,其结构见图 1(a)和 1(b)。

在图 1(a)中,Pred_Index 表的 IP_Attri_Vals_i 字段是元组 t 第 i 个不确定性谓词的属性值。

元组 t 满足的所有最小冗余确定性谓词以位图的形式压缩存储于 MRDP_Bitmap 字段。对于 n 个最小冗余确定性谓词 $pred_1,\cdots,pred_n$,其位图值是一个整数 bitmap。设 bitmap 的二进制形式为:$b_n b_{n-1}\cdots b_1$。其中:对于任意的 i,$1 \leqslant i \leqslant n$,$b_i=1$,if 元组 t 满足谓词 $pred_i$;否则 $b_i=0$。

Delete_Flag 是该谓词索引项的删除标志,false(0)代表未删除,true(1)代表该项已删除。

File_ID 是元组 t 所在磁盘缓存文件的序号,与该文件在第三级存储器中的压缩地址 Address 相对应(见图 1(b))。



ID:元组的序号
IP_Attri_Vals:不确定性谓词的属性值,IP_Attri_Val_i 是第 i 个不确定性谓词的属性值
MRDP_Bitmap:最小冗余确定性谓词的位图
Delete_Flag:删除标志
File_ID:与 MRDP_Bitmap 值对应的磁盘缓存文件的 ID

(a) 谓词索引表 Pred_Index

File_ID:与 MRDP_Bitmap 值对应的磁盘缓存文件的 ID
Address:磁盘缓存文件经压缩后在磁带库中的地址

(b) 压缩文件地址映射 Address_Map

图 1

通过 Pred_Index 和 Address_Map 在 File_ID 属性上的连接,可以得到以 IP_Attri_Vals 为不确定性谓词的属性值,同时最小冗余确定性谓词的 bitmap 为 MRDP_Bitmap,而且没有被删除的全部元组所在的压缩文件地址 $Address_{bitmap}$ 的集合。

在磁盘中保留三块缓存区(见图 2)。Compress_Buffer 作为压缩缓存区并分为若干块(每一块由一个磁盘文件组成,编号为 File_ID),不同确定性谓词的位图值 bitmap 对应不同的块。当其中一块压缩到第三级存储器后,为 File_ID 赋一个新的值。Decompress_Buffer 作为解压缓存区。在查询时,数据将首先在 Decompress_Buffer 中解压缩,然后再导入数据库。Update_Buffer 也按照位图值的不同划分为不同的缓存块,用于更新和删除。对于属于同一个压缩文件的元组,当其中的一些元组被删除后,其他元组将缓存到原位图值的更新缓存块中。如果有一些元组被更新,那么更新的元组将缓存到当前位图值的更新缓存块中,其余元组仍缓存到原位图值所对应的缓存块中。Update_Buffer 在适当的时候利用压缩算法重新压缩存储至第三级存储器。

2.3 第三级存储器中的存储结构

本文采用磁带库作为第三级存储器。根据文[4]的方法,将磁带库抽象为一个独立的文件系统,从而使磁带库具有更新、删除等功能。从第三级存储器中读取数据时,磁带定位时间在很大程度上影响着算法执行时间。为了减少数据传输量和磁带的磁头随机定位时间,在查询、删除和更新算法中将第二级存储器数据库的 TempTable 作为缓存,同时考虑了压缩文件在磁带库上的存储顺序。整个系统的结构见图 2。

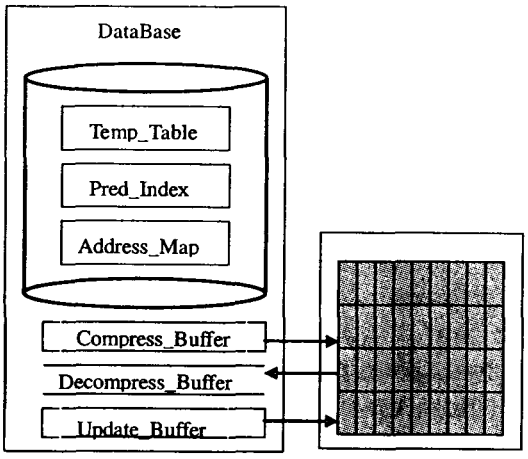


图 2　基于谓词索引的海量数据压缩存储结构

2.4　压缩存储算法

与存储结构对应的压缩算法描述如下：

对于海量关系 R 的每一条记录 t，首先计算 t 所满足的最小冗余确定性谓词的 bitmap 值，将 t 写入为该 bitmap 分配的待压缩磁盘文件缓存 $Compress\_Buffer_{bitmap}$，然后将 t 的不确定性谓词的属性值和 bitmap 作为索引项存储在二级存储器数据库的 Pred_Index 表中，同时在 Pred_Index 中记录 t 所在的磁盘文件缓存 Compress_Bufferbitmap 的序号 File_id-bitmap。如果文件缓存 $Compress\_Buffer_{bitmap}$ 的大小达到某一上限 SIZE，那么我们将采用给定的压缩算法 C，对 $Compress\_Buffer_{bitmap}$ 进行压缩。将压缩的文件写入第三级存储器，同时获得该压缩文件在第三级存储器上的地址 $Address_{bitmap}$。最后将 $File\_id_{bitmap}$ 和 $Address_{bitmap}$ 序对写入 Address_Map。其算法的形式化描述如下：

算法　Pred_Index_Compress

输入：海量关系 R，k 个不确定性谓词的属性，具有最小冗余的确定性谓词的集合 S，待压缩磁盘文件缓存大小上限 SIZE，使用的压缩算法 C

输出：海量关系在第三级存储器上的压缩文件集合以及在第二级存储器数据库中的谓词索引表 Pred_Index 和 Address_Map

1. 分配 Compress_Buffer，由 $2^{|S|}$ 个磁盘文件组成/*，每个位图值 bitmap 对应一个磁盘文件*/
2. for ∀ t ∈R
   2.1 由 S 计算 t 的最小冗余确定性谓词的位图值 bitmap
   2.2 把 t 写入为该 bitmap 所对应的磁盘文件缓存 $Compress\_Buffer_{bitmap}$，设其文件缓存序号为 $File\_id_{bitmap}$
   2.3 将 t 的不确定性谓词的属性值、位图值 bitmap、$File\_id_{bitmap}$ 以及 Delete_flag=false 插入 Pred_Index 表
      2.3.1 if（磁盘文件缓冲区 $Compress\_Buffer_{bitmap} \geqslant$ SIZE）
         2.3.1.1 利用压缩算法 C 压缩磁盘文件缓存 $Compress\_Buffer_{bitmap}$
         2.3.1.2 将压缩后的文件写入第三级存储器，获得其地址 $Address_{bitmap}$
         2.3.1.3 将 $File\_id_{bitmap}$ 和 $Address_{bitmap}$ 序对插入 Address_Map 表中

3　查询、更新和删除算法及分析

在数据操作（查询、更新和删除）时，需要由操作中的条件构造谓词索引。谓词索引如下构造：首先将数据操作的条件中所有谓词用不确定性谓词（IP）和最小冗余的确定性谓词（MRDP）代替，然后将 MRDP 表示成位图 bitmap 形式。于是，原有操作的条件就转化为由 IP 和 bitmap 组成的谓词索引。

3.1　查询算法

设 Q 为查询语句，C 是 Q 的查询条件，查询过程如下：首先将查询条件 C 转化为如下的谓词索引 C′∩(Delete_flag=false)，由于 C⊆C′∩(Delete_flag=false)，导致查询范围过大，因此查询需要分为两个阶段：第一阶段利用谓词索引 C′∩(Delete_flag=false)获得含有结果集的压缩文件所在磁带库上的地址集合 Address_Set，根据地址将压缩文件从磁带库依次导入磁盘的解压缓冲区 Decompress_Buffer 中，解压后导入数据库。在第二阶段利用查询条件 C 在数据库中进行二次查询，得出最终结果。在第一阶段，由于在数据库的 Temp_Table 中缓存了以压缩文件为单位而导入的元组，因此可以避免不必要的文件传输、解压和导入操作。同时，为了使磁带库在一遍扫描完成操作的同时减少数据传输，需要将获得的压缩文件地址 Addressbitmap 按照其在磁带库中的顺序进行排列。该算法的形式化描述如下：

算法 Query_Massive_Relation：

输入：查询语句 Q，其查询条件为 C，谓词索引表 Pred_Index 和 Address_Map，k 个不确定性谓词的属性，具有最小冗余的确定性谓词的集合 S，当前数据库中已经缓存在 Temp_Table 中元组所属压缩文件地址集合 X，解压缩算法 C

输出：查询结果

1 分配 Decompress_Buffer，由至少一个磁盘文件组成

2 将查询条件 C 转化为谓词索引 C′∩(Delete_flag=false)

3 根据谓词索引表 Pred_Index 和 Address_Map 的连接操作获得满足条件 C′∩(Delete_flag=false)的 Addressbitmap 的集合和元组 ID 的集合，分别记为 Address_Set、ID_Set

4 Address_Set= Address_Set -X，按照 Address_Set 在磁带库中的顺序进行排序

5 对于缓存在 Temp_Table 而且压缩文件地址属于 X 的元组直接用 Q 查询，输出查询结果。

6 for ∀ $Address_{bitmap}$ ∈ Address_Set
   6.1 将地址在 $Address_{bitmap}$ 的压缩文件从磁带库中导入磁盘解压缓存 Decompress_Buffer
   6.2 用 C 解压后，将 ID∈ ID_SET 的元组导入数据库的 Temp_Table
   6.3 利用 Q 重新查询数据库，输出查询结果

3.2　删除算法

对三级存储器中压缩元组的删除，将导致相应谓词索引的删除。设删除语句为 D，C 是删除条件，删除仍由两个阶段组成：第一阶段类似于查询算法的第一阶段。第二阶段，对 Temp_Table 进行删除 D 操作，获得被删除元组 ID 集合 ID_SET。将没有删除的元组原样导出数据库，缓存在其位图值 bitmap 所对应的 $Update\_Buffer_{bitmap}$ 的缓存块中，同时将谓词索引 ID∈ ID_SET 的索引项的 Delete_Flag 置为 true，标记该项已经删除。当累计删除个数超过某一上界 Max，就对数据库中的 Delete_Flag=true 的谓词索引进行真正的批量删除。其算法的形式化描述如下：

算法　Delete_Massive_Relation：

输入：删除语句 D，其删除条件为 C，谓词索引表 Pred_Index

和 Address_Map，k 个不确定性谓词的属性，具有最小冗余的确定性谓词的集合 S，当前数据库中已经缓存的元组所属压缩文件的地址集合 X，解压缩算法 C，全局计数器 counter，批量删除的上界 Max

输出：删除结果

1 分配 Decompress_Buffer，由至少一个磁盘文件组成

2 分配 Update_Buffer，由 $2^{|S|}$ 个磁盘文件组成

/ * 每一个位图值 bitmap 对应一个磁盘文件 * /

3 将删除条件 C 转化为谓词索引 $C' \cap$ (Delete_flag=false)

4 根据谓词索引表 Pred_Index 和 Address_Map 的连接操作获得满足条件 $C' \cap$ (Delete_flag=false) 的 Addressbitmap 集合和 ID 集合，分别记为 Address_Set、ID_Set

5 Address_Set= Address_Set - X，按照 Address_Set 在磁带库中的顺序进行排序

6 对于缓存在 Temp_Table、压缩文件地址属于 X 的元组直接用 D 删除，所删除元组的 ID 集合记为 ID_Set_1

7 for $\forall$ $Address_{bitmap} \in$ Address_Set

7.1 将地址在 $Address_{bitmap}$ 的压缩文件从磁带库导入磁盘解压缓存 Decompress_Buffer

7.2 用 C 解压后，将 ID$\in$ ID_SET 的元组导入数据库的 Temp_Table

7.3 对数据库中的元组进行删除操作 D，所删除元组的 ID 记为 ID_Set_2

7.4 将未删除的元组导出至其原 bitmap 所对应的 $Update\_Buffer_{bitmap}$

8 for $\forall$ ID $\in$ (ID_Set_1 $\cup$ ID_Set_2)

/ * 标记谓词索引项为已删除 * /

8.1 将 Pred_Index 中元组 ID 的 Delete_Flag 置为 true

8.2 counter = counter +1

/ * 记录标记删除的个数 * /

8.2.1 if (counter$\geqslant$Max)

8.2.1.1 批量删除 Delete_Flag= true 的所有谓词索引，counter =0

3.3 更新算法

对三级存储器中压缩元组的更新同样需要对谓词索引进行相应的更新。设更新语句为 U，C 是更新条件。同样分为两阶段：第一阶段类似于查询算法的第一阶段。在第二阶段对 Temp_Table 进行更新操作 U，获得更新元组的 ID 集合 ID_SET。将没有更新的元组原样导出数据库，缓存在其 bitmap 值所对应的 $Update\_Buffer_{bitmap}$ 的缓存块中；对于更新的元组，导出至其新 bitmap 所对应的 $Update\_Buffer_{bitmap}$ 缓存块中。同时将谓词索引 ID$\in$ ID_SET 的谓词索引项进行相应的更新，包括可能的不确定性谓词属性值、最小冗余确定性谓词的 bitmap 以及 File_ID 的更新。其算法的形式化描述如下：

算法 Update_Massive_Relation：

输入：更新语句 U，其更新条件为 C，谓词索引表 Pred_Index 和 Address_Map，k 个不确定性谓词的属性，具有最小冗余的确定性谓词的集合 S，当前数据库中已经缓存的元组所属压缩文件的地址集合 X，解压缩算法 C

输出：更新结果

1 分配 Decompress_Buffer，由至少一个磁盘文件组成

2 分配 Update_Buffer，由 $2^{|S|}$ 个磁盘文件组成/ * 每一个位图值 bitmap 对应一个磁盘文件 * /

3 将更新条件 C 转化为谓词索引 $C' \cap$ (Delete_flag=false)

4 由谓词索引表 Pred_Index 和 Address_Map 的连接操作获得满足条件 $C' \cap$ (Delete_flag=false) 的 $Address_{bitmap}$ 集合和 ID 集合，分别记为 Address_Set、ID_Set

5 Address_Set=Address_Set - X，按照 Address_Set 在磁带库中的顺序进行排序

6 对于缓存在 Temp_Table、压缩文件地址属于 X 的元组直接用 U 更新，所更新元组的 ID 集合记为 ID_Set_1

7 for $\forall$ $Address_{bitmap} \in$ Address_Set

7.1 将地址在 $Address_{bitmap}$ 的压缩文件从磁带库导入磁盘解压缓存 Decompress_Buffer

7.2 用 C 解压后，将 ID$\in$ ID_SET 的元组导入数据库的 Temp_Table

7.3 对数据库中的元组进行更新操作 U，所更新元组的 ID 记为 ID_Set_2

7.4 将未更新的元组导出至其原 bitmap 所对应的 $Update\_Buffer_{bitmap}$ 缓存块

7.5 将更新的元组导出至其新 bitmap 所对应的 $Update\_Buffer_{bitmap}$ 缓存块 $File\_id_{bitmap}$

8 for $\forall$ ID $\in$ (ID_Set_1 $\cup$ ID_Set_2)

/ * 更新谓词索引项 * /

8.1 重新填写 Pred_Index 中元组 ID 的 IP_Attri_Vals，重新计算元组 ID 的 bitmap，更新对应的 $File\_id_{bitmap}$

3.4 对查询、更新和删除中第一阶段查询效率的分析

由于对数据库操作条件中的谓词进行了索引，因此在查询、更新和删除操作的第一阶段，即压缩文件从第三级存储器向第二级存储器的导入过程中，被检索返回的压缩文件个数将大大减少。设 S 是最小冗余的确定性谓词集合，|S| 为该集合的大小，R 为不确定性谓词的集合，y 是在第三级存储器磁带库中存储的压缩文件总数。如果根据查询条件得到的谓词索引由 k 个最小冗余的确定性谓词和 d 个不确定性谓词构成，那么利用该谓词索引在第一阶段所返回的压缩文件个数 x 与 y 的比值满足如下定理：

定理 1

$$\alpha\left(\frac{2^{|S|-k}}{y} \leqslant \frac{x}{y} \leqslant \alpha\left(1-\frac{2^{|S|}-2^{|S|-k}}{y}\right)\right) \quad (0<1\leqslant 1)$$

证明：|S| 个最小冗余确定性谓词所确定的位图索引值 $bitmap \in [0, 2^{|S|})$，设满足第 i 个位图值 $bitmap_i$ 的压缩文件个数为 $N_i (N_i \geqslant 1)$，则在三级存储器中压缩文件的总数为 $y = \sum_{j=0}^{2^{|S|}-1} N_j$。由于 |S| 个最小冗余的确定性谓词只确定了 k 个，其余 $|S|-k$ 个没有确定，因此谓词索引中的 bitmap 有 $2^{|S|-k}$ 个不同的取值，这些不同的值记为集合 M。于是返回的压缩文件个数 x 与 y 的比值满足：

$$\frac{\sum_{j\in M}\min(N_i)}{\sum_{j=0}^{2^{|S|}-1} N_j} \leqslant \frac{x}{y} \leqslant \frac{\max(\sum_{i\in m} N_i)}{\sum_{j=0}^{2^{|S|}-1} N_i}$$

由于 $\min(N_i)=1$，因此 $\sum_{i\in M}\min(N_i)=2^{|S|-k}$。

而 $\max(\sum_{i\in M} N_i) = \sum_{j=0}^{2^{|S|}-1} N_j - \sum_{i\notin M}\min(N_i)$，$\sum_{i\notin M}\min(N_i) = \sum_{i\notin M} 1 = 2^{|S|} - 2^{|S|-k}$，所以

$$\frac{2^{|S|-k}}{\sum_{j=0}^{2^{|S|}-1} N_j} \leqslant \frac{x}{y} \leqslant 1-\frac{2^{|S|}-2^{|S|-t}}{\sum_{j=0}^{2^{|S|}-1} N_j} \qquad (1)$$

式(1)是只由 k 个最小冗余确定性谓词作为索引得到的结果。对于 d 个不确定性谓词，它会进一步限制满足条件的候选集大小，这一点将在后面的查询实验中得到验证。设其对候选集的影响因子为 $\alpha(0<\alpha\leqslant 1)$，于是式(1)改为：

$$\alpha\frac{2^{|S|-k}}{\sum_{j=0}^{2^{|S|}-1} N_j} \leqslant \frac{x}{y} \leqslant \alpha\left(1-\frac{2^{|S|}-2^{|S|-k}}{\sum_{j=0}^{2^{|S|}-1} N_j}\right)$$，得证。

4 实验结果及分析

实验数据：实验数据使用由 TPC-H Benchmark[11] 生成工具 dbgen 生成的 Lineitem 表。dbgen 将按照主键 L_OrderKey 和 L_LineNumber 值递增的顺序产生无偏斜的 Lineitem 记录。我们对压缩比的定义如下：压缩比＝(1－ 压缩后大小/压缩前大小)×100%

4.1 底层压缩算法 C 的选择

对于由 Lineitem 元组构成的不同大小的数据文件，分别采用自适应 Huffman(AH)、LZW 和二阶算术编码(AC2)进行压缩，比较压缩时间、解压时间和压缩比，通过比较得到针对 Lineitem 元组最优的底层压缩算法。实验结果见表 1。

由实验数据可以得出，对于 TCP-H Benchmark 中的 Lineitem 表，二阶算术编码(Arithmetic Coding order-2)的压缩比(高于 70%)明显优于其它两种方法，但其压缩和解压缩时间随文件大小的增加呈倍数增长。这归结于二阶算术编码在压缩和解压缩时要进行大量频繁的算术运算，占用了几乎 100%的 CPU。LZW 的压缩比虽低于二阶算术编码，但在实验中它的压缩比稳定在 59%以上，而且压缩和解压缩时间也是这三种算法中最好的。因此，底层压缩算法 C 取 LZW。

4.2 压缩比的比较

利用 dbgen 分别产生 0.35G(2999671 条记录)、0.7G(6001189 条记录)和 1.4G(12004754 条记录)的 Lineitem 表数据。根据磁盘缓存文件的大小 SIZE 的不同来比较压缩比。实验结果见表 2。表 2 说明 LZW 对不同数据量的压缩比是稳定的，这进一步验证了实验 1 的结果，从而使海量数据的压缩比稳定在 60%以上。

4.3 查询效率的比较

由于查询、更新和删除的时空开销都集中在第一阶段，因此我们只给出查询操作的实验结果。更新和删除操作与此类似。实验比较基于谓词索引的查询算法 Query_Massive_Relation(记为 A)和工业界采用的压缩存储查询算法(见引言，记为 B)。实验数据中，不确定性谓词的属性分别为 L_OrderKey 和 L_LineNumber(均为主键)，最小冗余的确定性谓词中涉及的属性包括：L_Discount、L_ShipDate 和 L_Quantity。设计如下三种查询：

Q_1：查询条件 C 中只有不确定谓词

C＝ L_OrderKey＜500

Q_2：查询条件 C 中只有确定性谓词

C＝(L_Discount≤0.02 and L_Quantity＜10

and L_ShipDate＜1995-9-9)

Q_3：查询条件 C 同时包含不确性谓词和确定性谓词

C＝(L_Discount≤0.02 and L_LineNumber＝2 and L_OrderKey＜100)

图 3 是分别将 0.35G 和 0.70G 的原始数据压缩后，在不同的查询语句 Q_1、Q_2、Q_3 下，A 与 B 查询时间的比值 A/B。该图说明我们的查询算法 A 明显优于算法 B，特别是在包含不确定性谓词的 Q_1 和 Q_3 中，这与我们在定理 1 中得到的结果吻合。这是因为以不确定性谓词的属性为主键，可以准确地确定满足条件的元组所在压缩文件的地址，而在 Q_2 中，由于最小冗余的确定性谓词 C_2 获得的查询结果是确定性谓词 C_1 的超集，因此查询会返回一些满足 C_2 但不满足 C_1 的压缩文件地址，从而造成 Q_2 的查询效率较低，但仍优于工业方法 B。

图 4 是分别将 0.35G 和 0.70G 的数据压缩后，在不同的查询语句 Q_1、Q_2、Q_3 下，算法 A 和 B 所解压的文件数目。可以看出算法 A 同样优于算法 B，这是因为工业算法 B 是全部解压，而算法 A 由于使用了谓词索引，所以只将相关的压缩文件从磁带库调入磁盘解压查询。

结论 本文提出了以谓词为索引的海量数据压缩存储结构，并基于该存储结构讨论了相应的数据操作算法。理论分析和实验结果表明，这种将第二级存储器和第三级存储器结合起来使用的存储结构既可以有效减小数据的存储空间又可以提高查询和数据操作的效率。如何进一步改进压缩存储结构以支持更多的数据库操作，例如 Join 操作等，是我们下一步的研究方向。

表 1 底层压缩算法 C 的选择

文件大小 (kb)	压缩算法	压缩时间 (s)	解压缩时间 (s)	压缩比 (%)
100	AH	0.135	0.150	36.0
	LZW	0.060	0.050	60.0
	AC2	0.691	0.661	72.0
200	AH	0.271	0.301	36.7
	LZW	0.121	0.090	59.8
	AC2	1.312	1.282	72.9
500	AH	0.640	0.731	37.6
	LZW	0.311	0.251	61.8
	AC2	3.295	3.195	74.1
1000	AH	1.292	1.432	36.9
	LZW	0.611	0.470	61.9
	AC2	6.489	6.330	74.6
2000	AH	2.543	2.844	37.0
	LZW	1.222	0.972	60.3
	AC2	12.969	12.588	75.0

表 2 海量关系压缩比的比较(压缩算法 C＝LZW)

Lineitem 原始数据大小	SIZE (kB)	压缩后数据大小 (GB)	压缩比 (%)
0.35GB	100	0.314	61.7
	200	0.132	62.3
	500	0.131	62.6
0.7GB	100	0.268	61.7
	200	0.265	62.1
	500	0.264	62.3
1.4GB	100	0.550	60.7
	200	0.546	61.0
	500	0.540	61.4

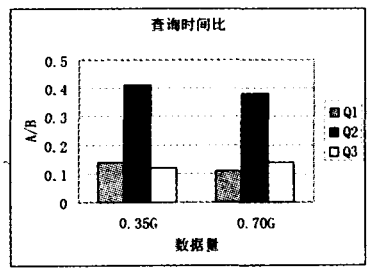


图 3 查询算法 A 与 B 查询时间比

(下转第 159 页)

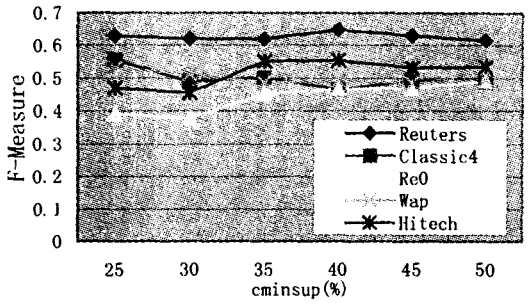


图 2 簇最小支持度的影响

实验比较：表 7 给出了算法平均精度的比较结果，表 8 则是算法最高精度的比较结果。

R-means 的平均精度的获取是通过固定 *gminsup* 为一个合适的值，然后通过变化一组其余参数，最后将得到的 F-Measure 值取平均得到。而最高精度则是一组实验中获得的最高 F-Measure 值。

表 7 平均精度比较

数据集	整体 F-Measure(平均)				
	R-means	FIHC	UPGMA	Bi-kmeans	HFTC
Classic4	0.50	0.54	—	0.44	0.61
Hitech	0.52	0.42	0.38	0.37	0.37
Re0	0.44	0.45	0.40	0.34	0.43
Reuters	0.63	0.60	—	0.39	0.49
Wap	0.50	0.52	0.51	.45	0.35

表 8 最高精度比较

数据集	整体 F-Measure(平均)				
	R-means	FIHC	UPGMA	Bi-kmeans	HFTC
Classic4	0.56	0.62	—	0.59	0.61
Hitech	0.55	0.45	0.47	0.54	0.37
Re0	0.49	0.53	0.47	0.38	0.43
Reuters	0.65	0.61	—	0.48	0.49
Wap	0.53	0.57	0.59	0.57	0.35

由表 7 和表 8 可以看出，*R*-means 无论是平均精度还是最高精度总是能达到或接近最优结果。

3.4 性能

为测试算法性能，我们选择五个数据集中最大的数据集 *Reuters* 来进行该项测试，测试平台是系统为 Windows2000 的 Celeron 2G 的 PC 机。当 gminsup 设置为 10%时，一组实验表明，尽管该数据集包含超过 8000 个文档，算法的平均运行时间仅为 53 秒，并且算法所需的平均内存量也仅为 5.4 兆。

结论 本文将传统 *k*-means 算法和关联文本聚类技术相结合，提出了一种新颖的文本聚类方法。通过在多个现实数据集上的实验证明，该算法可以取得很高的精度，并具有良好的性能。

参 考 文 献

1 Fung B C M, Wang K, Ester M. Hierarchical Document Clustering Using Frequent Itemsets. In: Proc. of the 2003 SIAM Intl. Conf. on Data Mining(SIAM'03)
2 Agrawal R, Srikant R. Fast algorithms for mining association rules. VLDB-94, 1994
3 MacQueen. Some methods for classification and analysis of multivariate observations. In: Proc. 5th Berkeley Symp., I, 1967. 281～297
4 Beil F, Ester M, Xu X. Frequent term-based text clustering. In: Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD)'2002, Edmonton, Alberta, Canada, 2002
5 Dubes R C, Jain A K. Algorithms for Clustering Data. Prentice Hall College Div, Englewood Cliffs, NJ, March 1998
6 Karypis G. Cluto 2.0 clustering toolkit. April 2002. http://www-users.cs.umn.edu/~karypis/cluto/
7 Steinbach M, Karypis G, Kumar V. A comparison of document clustering techniques. In: KDD Workshop on Text Mining'00, 2000
8 Mitchell T. Machine Learning. McGraw Hill, 1997
9 Text REtrival Conference TIPSTER, 1999. http://trec.nist.gov/
10 Han E-H, Boly D, Gini M, et al. WebACE: A Web Agent for Document Categorization and Exploration. In: Proc. Agents 98
11 http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html
12 van Rijsbergen C J. Information Retrieval. Dept. of Computer Science, University of Glasgow, Butterworth, London, 2 edition, 1979
13 Classic. ftp://ftp.cs.cornell.edu/pub/smart/

(上接第 90 页)

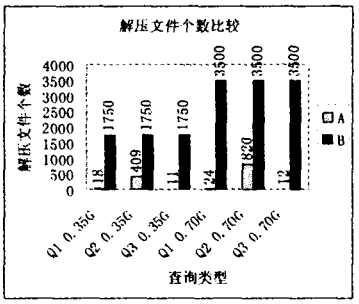


图 4 查询算法 A 与 B 解压文件个数比较(SIZE=200kb)

参 考 文 献

1 Luo J Z, Li J Z, Wang H Z, et al. The Compression of Massive Offline Relations. WAIM, 2004. 634～639
2 Goldstein J, Ramakrishnan R, Shaft U. Compressing Relations and Indexes. ICDE, 1998. 370～379
3 Westmann T, Kossmann D, Helmer S, et al. The Implementation and Performance of Compressed Databases. SIGMOD, 2000, 29(3): 55～67
4 张艳秋，李建中. 海量信息存储系统中一种基于磁带的文件系统. 高技术通讯，2003，13(2)
5 Nelson M, Gailly J-l. The Data Compression 2nd Edition. New York: M&T Books, 1995
6 Poess M, Potapo D. Data compression in oracle. VLDB, 2003. 937～947
7 Furtado P, Madeira H. FCompress: A New Technique for Queriable Compression of Facts and Datacubes. IDEAS, 2000. 197～206
8 Gao H, Li J. Cube Algorithms on Very Large Compressed Data Warehouses. Journal of Software, 2001, 12(6)
9 Li J Z, Srivastava J. Efficient Aggregation Algorithms for Compressed Data Warehouses. TKDE, 2002, 14(3): 515～529
10 Margaritis D, Faloutsos C, Thruns S. NetCube: A Scalable Tool for Fast Data Mining and Compression. VLDB, 2001. 311～320
11 Transaction Processing Performance Council. TPC BENCHMARK H (Decision Support) StandardSpecification. http://www.tpc.org/tpch/default.asp
12 Li J, Rotem D, Srivastava J. Aggregation Algorithms for Very Large Compressed Data Warehouses. VLDB, 1999. 651～662