基于分组序号的聚集算法*

冯建华[1+], 蒋旭东[1], 孟宪虎[2]

[1](清华大学 计算机科学与技术系,北京 100084)

[2](运城高等专科学校 计算机系,山西 运城 044000)

An Aggregation Algorithm Based on Group Numbers

FENG Jian-Hua[1+], JIANG Xu-Dong[1], MENG Xian-Hu[2]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[2](Department of Computer, Yuncheng Advanced Training College, Yuncheng 044000, China)

+ Corresponding author: Phn: 86-10-62789150, E-mail: fengjh@tsinghua.edu.cn

http://www.tsinghua.edu.cn

Received 2001-10-22; Accepted 2002-04-22

Feng JH, Jiang XD, Meng XH. An aggregation algorithm based on group numbers. *Journal of Software*, 2003,14(2):222~229.

Abstract: OLAP (online analytical processing) queries are complex. When implemented in SQL (structured query language), they usually involve multi-table join and aggregate operations. As a result, how to improve the performance of the multi-table join and aggregate operations becomes a key issue for ROLAP (relational OLAP) query evaluation. To solve this problem, an aggregation algorithm based on group numbers named MuGA (group number based aggregation with multi-table join) is proposed in this paper. By taking the characteristics of star schema into consideration, the algorithm combines the aggregation operation with the novel multi-table join algorithm, Mjoin (multi-table join), and replaces the sorting and hashing method by computed group numbers in aggregation computing. As a result, the algorithm can not only reduce the CPU time, but also reduce the disk I/Os for OLAP queries. As illustrated by the experiments, the performance of the algorithm MuGA is superior to original aggregation methods and the new sorting based method for aggregation.

Key words: data warehouse; OLAP(online analytical processing); multi-table join; aggregation query

摘 要: 联机分析处理 OLAP(online analytical processing)查询作为一种复杂查询,当使用 SQL(structured query language)语句来表述时,通常都包含多表连接和分组聚集操作,因此提高多表连接和分组聚集计算的性能就成为 ROLAP(relational OLAP)查询处理的关键问题.提出一种基于分组序号的聚集算法 MuGA(group number based aggregation with multi-table join),该方法充分考虑数据仓库星型模式的特点,将聚集操作和新的多表连接算法 MJoin(multi-table join)相结合,使用分组序号进行分组聚集计算,代替通常的排序或者哈希计算,从而有效地减少 CPU 运算以及磁盘存取的开销.算法的实验数据表明,提出的 MuGA 算法与传统的关系数据库聚集查询

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No.G1998030414 (国家重点基础研究发展规划(973))

第一作者简介: 冯建华(1967－),男,山西运城人,副教授,主要研究领域为数据库,数据仓库,WWW 环境下的信息处理.

处理方法以及改进后的基于排序的聚集算法相比,性能都有显著提高.

关键词: 数据仓库;联机分析处理;多表连接;聚集查询
中图法分类号: TP311 文献标识码: A

联机分析处理是数据仓库系统最主要的一种应用,OLAP 查询一般都是涉及大量数据的即席复杂查询[1].目前主要有两种方式可用于实现 OLAP 查询.一种是将数据存储在多维数组中,在多维数组的基础上来实现各种 OLAP 操作,这种方式称为 MOLAP(multi-dimensional OLAP).MOLAP 能够提供较快的查询响应速度,但是在进行数据更新时通常需要重构多维数组,额外开销比较大.另外,在多维数组非常稀疏时,即使使用数据压缩技术也可能浪费大量的磁盘存储空间.由于 MOLAP 存在这些缺点,在某些情况下仍然需要使用传统的关系数据库,基于星型模式来存储多维数据,然后通过多表连接、分组聚集计算等操作来实现 OLAP 查询,这种方式称为 ROLAP(relational OLAP).

针对传统的 OLTP(online transaction processing)应用,关系数据库系统对各种数据操作的查询处理进行了优化,但是由于没有充分考虑数据仓库中数据及其应用的特点,对多表连接和聚集计算的优化还很不够.一般来说,数据仓库中存储着大量的多维历史数据,并且数据很少需要修改,主要应用于多维数据分析、决策支持系统、数据挖掘等.针对这些特点,目前已经有一些新的方法可用于提高 ROLAP 查询的响应速度,如新的索引技术[2]、实物化视图技术[3~5]、CUBE 操作及其实现算法[6,7]、联机聚集[8]和近似查询[9]等等.对于 ROLAP 系统,OLAP 查询一般都涉及多表连接和分组聚集操作,提高这些操作的性能成为提高 OLAP 响应速度的关键.但是目前关于分组聚集计算,一些研究成果主要都集中于聚集查询的并行处理[10]以及如何针对聚集操作来进行查询优化[11],而有关如何针对数据仓库中数据及其应用的特点来设计新的聚集处理算法方面的论文还很少.

目前,分组聚集计算主要有基于排序和基于哈希两种方式[12],这两种方法的算法复杂度基本相同,商用的关系数据库系统大多采用基于排序的聚集计算方法.为了提高聚集计算的性能,文献[13]提出了一种新的基于排序的聚集算法 MuSA[13],该算法充分考虑了数据仓库中星型模式数据组织的特点,使用 MJoin 算法[14]同时进行多表连接,并且排序时采用关键字映射技术对排序关键字进行压缩,通过提高排序速度显著地提高了聚集计算的性能.经过进一步的研究发现,还可以利用星型结构的特点,首先计算出各个维表中每个分组字段取值对应的分组序号,然后再利用分组序号直接进行聚集计算来代替通常的排序或者哈希计算,不但有效地减少了 CPU 运算的开销,还减少了磁盘存取开销.据此,我们提出一种新的基于分组序号的聚集算法 MuGA.

本文第 1.1 节给出数据仓库星型模式的一个实例,第 1.2 节简要描述适合星型模式的多表连接算法 MJoin(聚集查询处理时将使用该多表连接算法),第 1.3 节详细讨论基于分组序号的聚集算法 MuGA,第 1.4 节对 MuGA 算法进行简要的分析,给出算法的复杂性度量.随后在第 2 节给出 MuGA 算法的实现环境和实现方式,并给出 MuGA 算法与传统的聚集查询处理以及基于排序的聚集算法 MuSA 进行比较的实验结果.第 3 节是总结,并指出为提高 OLAP 查询处理的性能需要进一步研究的问题.

1 聚集查询算法 MuGA

1.1 星型模式

为了能够直观地表示数据的多维本质,减少数据冗余,便于进行多维数据分析,数据仓库中的数据通常按照星型模式(或者雪花模式)进行组织[1].下面是一个星型结构实例:

事实表: Sales(TimeID,RegionID,ProductID,SaleNum,income)

维表: DimTime(TimeID,year,month,day)

DimRegion(RegionID,area,province,city)

DimProduct(ProductID,type,ProductName,price)

该星型结构对应于数据仓库的销售主题,由一个事实表 Sales 和 3 个维表 DimTime,DimRegion,DimProduct 共同组成(3 个维表分别给出了销售数据对应的时间、地域和产品).其中 3 个维表的主码分别为 TimeID,

RegionID,ProductID,并且作为事实表 Sales 的外码将事实表和维表相关联.一般来说,事实表中的记录数远远大于维表中的记录数.

当按照星型结构组织数据时,数据仓库的各种应用提交的 OLAP 查询一般都涉及分组聚集计算和多表连接计算.如下面的查询实例 Q1 要求按年、地区和产品类型得到总销售数量和总的销售收入,这是一个比较典型的 OLAP 聚集查询:

```
SELECT year,area,type,sum(SaleNum),sum(income)
FROM DimTime,DimRegion,DimProduct,Sales
WHERE DimTime.TimeID=Sales.TimeID AND
        DimRegion.RegionID=Sales.RegionID AND
        DimProduct.ProductID=Sales.ProductID
GROUP BY year,area,type
```

1.2 多表连接算法MJoin

在传统的关系数据库系统中,对于多表连接,每次都选择其中的两个表进行连接运算.能够进行的查询优化也仅仅是基于查询代价选择连接的处理方法(如嵌套循环、基于哈希的连接、基于排序的连接等等)以及选择关系表两两进行连接的次序[12].在数据仓库环境下,当数据按照星型结构进行组织时,经常会遇到涉及多个表的连接操作,由于维表之间的连接相当于笛卡尔积,而事实表和维表之间的连接通常并不能减少记录数目,此时仍使用传统的多表连接处理方法,效率就很低.

为了提高星型连接查询的性能,本文提出了一种新的多表连接算法 MJoin[14].与传统关系数据库管理系统的多表连接查询处理相比,该算法充分考虑了数据仓库数据本身和多表连接的特点,不需要借助额外的索引,就可使得多表连接查询的性能有明显改善.

考察一下星型结构数据组织的特点,一般维表中的记录数都很有限,而事实表中的记录数却很多,与维表相比通常会相差好几个数量级,并且表之间的连接都通过一个中心表,即事实表来进行.因此可以考虑一次同时进行多个表的连接操作,也就是首先将维表中的记录读出,存入内存中的哈希表、排序表,或者其他可按连接属性值进行检索的数据结构,然后顺次读事实表中的记录,对应于读出的每条记录,根据连接属性查找内存中根据各个维表生成的有序表,得到相应的元组,最后再合并这些元组完成连接操作.

MJoin 算法充分利用星型结构多表连接的特点,同时进行多个表的连接操作,与传统方法相比,能够显著减少磁盘的 I/O 操作,从而提高多表连接的性能.

1.3 聚集查询算法MuGA

目前的聚集计算方法主要有基于排序和基于哈希两种.文献[13]提出了一种新的基于排序的聚集算法 MuSA,该方法考虑了星型结构的特点,使用各维内部的分组序号代替分组字段,组合产生排序关键字,从而显著提高了聚集计算的性能.经过进一步的研究发现,可以更加有效地利用各维内部的分组序号来提高聚集计算的性能.这里给出一种新的基于分组序号的聚集算法 MuGA,该方法使用各维内的分组序号组合得到分组聚集的分组序号,然后根据这个分组序号直接进行分组计算,这样相比 MuSA 算法,避免了开销很大的排序操作,提高了聚集算法的性能.

对于聚集查询 Q1,其分组字段是 year,area 和 type,分别属于维 DimTime,DimRegion 和 DimProduct.首先通过各维的分组字段,得到维表中每条记录在维内的分组序号.

下面以维表 DimTime 为例来说明如何对维表进行处理,从而得到维表中每条记录的分组序号.首先获得维表 DimTime 分组字段 year 的所有当前取值,并按其值进行排序,给出每个取值的分组序号,见表 1.

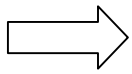
Table 1 Year is the group field of dimensional table DimTime

表 1 维表 DimTime 的分组字段 Year

Year	1997	1998	1999
Group NO	0	1	2

在得到每个分组字段值所对应的分组序号以后,就可以给出维表 DimTime 的每条记录所对应的分组序号 GroupNO,如图 1 所示.此外,对于维表 DimTime 还可以得到对维表 DimTime 按字段 Year 进行分组的分组数 $Groups_{year}$,也就是字段 Year 在数据仓库中当前的不同取值数目,$Groups_{year}$=3.

TimeID	Year	Month	Day
1	1997	1	1
2	1997	1	2
…	…	…	…
366	1998	1	1
367	1998	1	2
368	1998	1	3
…	…	…	…



TimeID	Group NO
1	0
2	0
…	…
366	1
367	1
368	1
…	…

Fig.1 The group number for each record of dimensional table DimTime

图 1 维表 DimTime 的每个记录的分组序号

对于维表 DimRegion,DimProduct 也将进行与 DimTime 同样的处理.

聚集查询处理时,对于事实表的每条记录,通过连接字段查询内存中的各个维表(经过上述处理后的维表),就可以得到对应各个维表内部的分组序号,假设分别为 $GroupNO_{year}$,$GroupNO_{area}$ 和 $GroupNO_{type}$,则将这些分组序号组合起来就可以得到进行分组聚集计算的分组序号.计算方法如下:

$$GroupNO=(GroupNO_{year}*Groups_{area}+GroupNO_{area})*Groups_{type}+GroupNO_{type}.$$

假设聚集结果的分组数目为各个维分组数目的乘积(实际分组数目只会比这个值小),对于 Q1 也就是 $Groups_{year}*Groups_{area}*Groups_{type}$.如果分组聚集结果记录能够全部放入内存(对于分组聚集的层次较高的查询,如 Q1,由于聚集查询结果的分组数目较少,该条件比较容易满足,而对于分组聚集层次较低的查询,由于结果的分组数目较多,将很可能超出可用内存的范围),那么分组聚集计算可以直接在内存中进行,效率将很高.否则,就需要根据分组序号对所有记录进行划分并分别写入磁盘,然后针对每个分片在内存中进行分组聚集计算.

下面给出 MuGA 算法的一般描述,这里先假设内存中可以放下所有的分组结果记录(随后再讨论更一般的情况),为描述简单起见,假设聚集计算仅为聚集字段 A 的求和运算.

算法. MuGA.

输入:事实表 FT,维表 $DT_1,\ldots,DT_m$;

聚集查询 Q;

进行分组聚集计算的缓冲区 *GroupBuf*.

(1) 首先对 Q 进行分析,得到各分组字段 $GA_1,\ldots,GA_m$,聚集字段 $Sum(A)$,此外还生成一系列存取维表数据的单表查询 $Q_1,\ldots,Q_m$,其中 Q_i 为对维表 DT_i 的简单查询,仅包含原查询 Q 中与维表 DT_i 有关的查询条件和相关字段(包括连接字段和分组字段);

(2) For i=1 to m

(2.1) 提交查询 Q_i,得到维表 DT_i 中满足查询条件的所有记录;

(2.2) 取出分组字段 GA_i 的所有取值并进行排序,然后给出每个分组字段值对应的分组序号 $GroupNO_i$,并同时得到分组数目 $Groups_i$;

(2.3) 得到该维表每条记录对应的分组序号;

End For

(3) 根据分组数目 GROUPS 等于各维分组数目的乘积,初始化缓冲区 *GroupBuf*;

(4) While 顺序扫描事实表 *FT*,对应于得到的每条记录 *R*:

(4.1) $GroupNO$=0;

(4.2) For i=1 to m

(4.2.1) 通过连接字段 ID_i,查找维表 DT_i 排序后的记录列表,得到分组序号 $GroupNO_i$;

(4.2.2) If i>1

$$GroupNO=GroupNO*Groups_i;$$

(4.2.3) $GroupNO=GroupNO+GroupNO_i$;

End For

(4.3) 生成连接后的记录(*GroupNO*,*A*);
(4.4) 根据 *GroupNO* 的值,定位缓冲区 *GroupBuf* 的第 *GroupNO* 条记录(*GroupA*);
(4.5) 计算 *GroupA*=*GroupA*+*A*;
(5) For *i*=0 to *GROUPS*–1
(5.1) 得到缓冲区 *GroupBuf* 的第 *i*+1 条记录(*GroupA*);
(5.2) If *GroupA*<>0
(5.2.1) *GroupNO*=*i*;
(5.2.2) For *j*=*m* to 1

$$GroupNO_j=GroupNO \bmod Groups_j;$$

根据 $GroupNO_j$ 查找位于内存中的相应维表,得到分组字段值 GA_j;
End For
(5.2.3) 将 $GA_1,\ldots,GA_m$ 和 *GroupA* 作为分组聚集计算的一条结果记录写入磁盘;
End If
End For

请注意,在使用 MuGA 算法进行分组聚集计算时,*GroupBuf* 中的某个结果分组可能并不存在,步骤(5)通过判断聚集结果 *GroupA* 是否为 0 来决定是否需要输出该结果分组.

当结果分组数目很多,无法将所有结果分组都放入内存时,需要对输入的记录进行划分.假设聚集计算可用的内存空间为 *M*,分组结果记录的长度为 *GR*,估计出的总分组数目为 *Groups*,则进行划分时的主要方法及步骤如下:

(1) $Groups_{memory}=M/GR$; /*计算出内存所能够容纳的分组数目 $Groups_{memory}$*/

(2) $Partition=(Groups-1)/Groups_{memory}+1$; /*决定分片的数目*/

(3) $PartioinNO=GroupNO/Groups_{memory}$; /*对于连接后产生的记录(*GroupNO*,*A*),由 *GroupNO* 值,确定该记录所属的分片*/

对所有记录都按照上面的步骤(3)进行划分并将分片写入磁盘后,再顺序处理每个分片,也就是读入每个分片的所有记录,然后按上面给出的聚集算法,进行分组聚集计算并将结果写入磁盘.为了进一步提高划分时内存的利用效率,还可以在划分的同时对第 1 个分片进行聚集计算.

MuGA 算法使用各个维表内部的分组序号组合成的分组序号来进行分组聚集计算,虽然增加了一些额外操作,包括分组序号的生成以及聚集计算完成后由分组序号反算每个维表内的分组序号并得到分组字段值.但是由于上述计算仅涉及简单的整数算术运算,增加的额外开销并不是很大.另一方面,通过使用分组序号代替开销很大的排序操作,使得 MuGA 算法与基于排序的聚集算法相比,性能显著提高.

此外,在实现 MuGA 算法的过程中,我们发现读事实表数据以及读写暂存的中间数据使用的都是顺序读写方式.为了提高磁盘存取性能,还使用了双缓存区读写技术.也就是说,程序具有两个同样大小的读写缓冲区,当 CPU 在处理其中一个缓存区中的数据时,另一个缓冲区仍与磁盘交换数据,两个缓冲区轮流,从而使得磁盘读写操作和 CPU 计算能够近似地并行执行.

1.4 MuGA算法的分析

为便于进行算法分析,首先假设事实表记录数为 *n*,存储于磁盘上所需要的物理磁盘块数为 *fb*,维表的数目为 *d*.另外还假设聚集结果分组数目为 *g*,其存储于磁盘上所需要的物理磁盘块数为 *fg*.由于事实表记录数目远远大于各维表记录数目,因此在下面的算法分析中将忽略读维表数据以及在开始时对维表进行各种预处理的开销.

如果聚集查询的结果分组能够全部放入内存,MuGA 算法在进行聚集计算时,无论事实表记录数目有多少,只需要顺序扫描一遍事实表(原始数据),并且计算过程中也不需要往磁盘上暂存任何中间结果,这样磁盘 I/O 开销达到了最小.而从 CPU 计算角度来说,每条事实表记录只需要处理一次(查找内存中相应的维表,计算得到该记录相应的分组序号,完成聚集计算),并需要对每条分组结果记录处理一次(根据生成的分组序号反算各维内部的分组序号,并查找内存中相应维表得到分组字段值).综上所述,MuGA 算法总共需要 *fb*+*fg* 次磁盘读写操作,

并需要 $n*d+g*d$ 次查找,n 次分组序号计算,g 次分组序号反算,n 次聚集计算.

当内存中无法存储所有的聚集结果分组时,就需要首先对事实表记录进行划分,然后再针对各个分片进行聚集计算.由于划分是根据分组序号进行的,并且保证划分后的每个分片都可以在内存中完成聚集计算,而不会像一般的哈希划分方法可能导致再次划分的发生,因此只需要对事实表数据进行三遍磁盘读写操作(首先读事实表,与维表进行连接;然后进行划分并将划分段写入磁盘;最后再顺序读取划分段并进行分组聚集计算),并且其中两遍读写是针对连接后的记录的,形式如($GroupNO$,A),与形式为(ID_1,…,ID_m,A)的原始事实表记录相比,长度一般要短.如果假设原始事实表记录长度是连接后记录长度的 k 倍,那么 MuGA 算法所需要的总的磁盘读写次数为 $fb+2fb/k+fg$.最后,从 CPU 计算的角度来说,由于每条事实表记录仍然只需要处理一次,所以 MuGA 仍只需要 $n*d+g*d$ 次查找,n 次分组序号计算,g 次分组序号反算以及 n 次聚集计算.另外,由于划分阶段的存在还需要加上 n 次划分计算的开销.

如果使用基于排序的聚集算法进行聚集计算,当事实表数目很大,无法在内存中对所有连接后的记录进行排序时,需要额外的归并排序阶段来进行排序.此时聚集算法的磁盘读写次数与归并排序的趟数直接有关,归并排序阶段至少需要 $fb+2fb/k$ 次磁盘读写操作(假设只需一趟归并排序,事实表记录数目过多时可能需要多趟归并排序),那么总的磁盘读写次数至少为 $fb+2fb/k+fg$ 次.由此基于排序的聚集算法的磁盘读写开销通常要比 MuGA 大.在排序过程中,CPU 的主要计算是比较操作以及内存数据的移动,总的数量级是 $n\lg n$.再加上 $n*d+g*d$ 次查找,n 次排序关键字生成计算以及 n 次聚集计算的开销,其 CPU 计算的总开销也要比 MuGA 算法略大.

由上面的算法分析可以知道,MuGA 算法与基于排序的聚集算法相比,在算法复杂性度量方面,包括磁盘存取开销和 CPU 计算开销两方面都有优势.下一节的实验结果也将进一步证实这个结论.

2 算法实现与实验结果分析

2.1 算法实现环境和实现方法

为进行算法实验结果比较,我们还分别实现了传统的基于排序的聚集算法 SortAgg(多表连接时使用了新的多表连接算法 MJoin),面向数据仓库进行了改进的基于排序的聚集算法 MuSA[13]以及基于分组序号的聚集算法 MuGA,并进行了算法实验.

实验用的星型模式的各表结构和查询 $Q1$ 如第 1.1 节所给出,另有聚集查询 $Q2$,如下所示.

```
SELECT    year,city,ProductName,sum(SaleNum),sum(income)
FROM      DimTime,DimRegion,DimProduct,Sales
WHERE     DimTime.TimeID=Sales.TimeID AND
          DimRegion.RegionID=Sales.RegionID AND
          DimProduct.ProductID=Sales.ProductID
GROUP BY  year,city,ProductName
```

该查询要求按年份、城市和产品得到总的销售数量和总销售收入.一般来说,查询 $Q1$ 的结果分组数目较少,而 $Q2$ 的结果分组数目较多.

星型结构中维表 DimTime 表的记录数为 1 080,DimRegion 表的记录数为 360,DimProduct 表的记录数为 500,事实表 Sales 表的记录数在 10 万以上,满足条件|Sales|>>|DimTime|,|Sales|>>|DimRegion|,|Sales|>>|DimProduct|.

2.2 实验结果分析

首先将算法 MuGA 与传统的基于排序的聚集算法 SortAgg 以及改进后的基于排序的聚集算法 MuSA 进行比较,实验所用环境为 Sun Sparc20,32M 主存.实验结果见表 2 和表 3,查询响应时间以秒为单位.

表 2 给出了聚集查询 $Q1$ 的实验结果.$Q1$ 的结果分组数目为 900.从表中的实验数据可以知道,MuGA 算法的加速比为传统的 SortAgg 算法的 10 倍左右,并且随着事实表记录数的增加而逐渐增大.此外,与 MuSA 算法相比其性能也略有提高.

表 3 给出了聚集查询 $Q2$ 的实验结果.$Q2$ 的估计分组数为 540 000,实际分组数目和事实表 Sales 的记录数有关.从表中的实验数据可以知道,MuGA 算法与 SortAgg 算法相比,加速比约为数十倍,并且随着事实表记录数的增加而逐渐增大,与 MuSA 算法相比,其性能也有所提高.

综上可知,MuGA 算法与传统的聚集查询处理算法 SortAgg 相比,对于各种分组聚集查询,查询性能都有显著提高,与改进的聚集算法 MuSA 相比,其性能也略有提高,但由于事实表记录数目不是很多,相比改进后的基于排序的聚集算法,MuSA 提高幅度不大,这与第 1.4 节中算法分析的结果相一致.

Table 2 The performance comparison of three different algorithms for query $Q1$

表 2 查询 $Q1$ 的 3 种算法的性能比较

The record number of Sales(s)	100 000	200 000	400 000
MuSA	3.0	5.1	7.9
MuGA	2.6	4.5	7.2
SortAgg	15.7	41.5	87.3

Table 3 The performance comparison of three different algorithms for query $Q2$

表 3 查询 $Q2$ 的 3 种算法的性能比较

The record number of Sales(s)	100 000	200 000	400 000
MuSA	4.5	8.0	16.3
MuGA	4.2	7.5	14.8
SortAgg	76.6	205.3	506.2

下面,我们增加事实表的记录数目(达到百万级),将基于分组序号的聚集算法 MuGA 与改进后的基于排序的聚集算法 MuSA 进行比较(使用聚集查询 $Q2$).实验环境为 Sun Ultra30,128M 主存,结果如图 2 所示.从图中的数据可以看出,MuGA 算法与算法 MuSA 相比其性能有显著提高,特别是随着记录数目的增多,性能提高的幅度也越来越大,这和前面对 MuGA 算法的分析是相一致的.很明显,MuGA 算法的 CPU 计算的复杂度为 $o(n)$,而基于排序的聚集算法的复杂度主要依赖于排序过程,近似为 $o(n\lg n)$,显然,随着事实表数目的增多,MuGA 算法的性能提高的也应越多.另外,随着记录数目的增加,可能会导致增加额外的归并排序趟数,使得 MuSA 算法的磁盘存取开销显著增加,最终也会造成 MuSA 算法的性能下降很快.

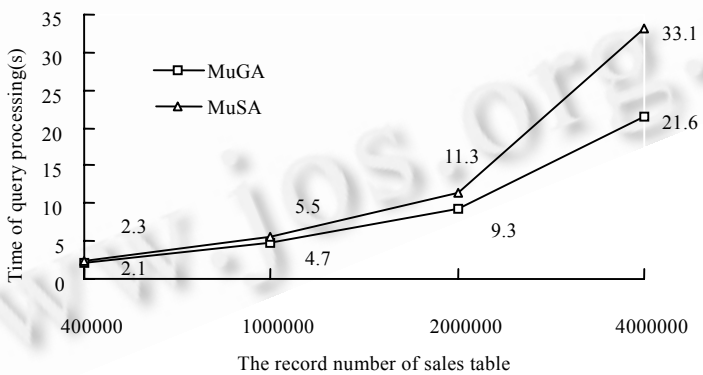


Fig.2 The performance comparison between MuGA and MuSA for query $Q2$

图 2 基于查询 $Q2$ 的算法 MuGA 和算法 MuSA 的性能比较

3 总 结

本文提出了一种新的基于分组序号的聚集算法 MuGA.该方法充分考虑了数据仓库的数据组织特点,将多表连接和聚集计算结合起来,使用算术运算计算出分组序号代替通常的排序或者哈希计算来进行分组聚集计算,从而有效地减少了 CPU 运算的开销,同时减少了磁盘存取开销.算法实验结果表明,与传统的基于排序的聚

集查询处理算法 SortAgg 以及改进后的基于排序的聚集算法 MuSA 相比,MuGA 算法的性能都有显著提高,并且随着事实表数目的增大,提高幅度也越来越大.

要实现完整的 OLAP 查询处理解决方案,在数据量很大时,仅仅依靠快速的聚集查询算法还很不够,此时需要结合其他技术,如利用实物化视图改写查询、并行计算、近似查询处理、联机聚集等技术来满足用户的不同需求.我们将继续结合各种数据仓库和 OLAP 查询处理技术,包括数据仓库的物理存储组织、并行查询处理和联机查询处理等方面,致力于提供完整的数据仓库和 OLAP 查询处理解决方案.

References:

[1] Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 1997,26(1):65~74.

[2] O'Neil P, Quass D. Improved query performance with variant indexes. ACM SIGMOD Record, 1997,26(2):38~49.

[3] Srivastava D, Dar S, Jagadish HV, Levy AY. Answering queries with aggregation using views. In: Vijayaraman TM, ed. Proceedings of the 22ed International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1996. 318~329.

[4] Gupta A, Harinarayan V, Quass D. Aggregate-Query processing in data warehousing environments. In: Umeshwar D, ed. Proceedings of the 21st International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1995. 358~369.

[5] Jiang XD, Zhou LZ. Implementation of OLAP query using materialized views. Journal of Lanzhou University (Natural Science), 1999,35(8):242~247 (in Chinese with English Abstract).

[6] Gray J, Bosworth A, Layman A, Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: Stanley YWS, ed. Proceedings of the 12th International Conference on Data Engineering. New Orleans: IEEE Computer Society, 1996. 152~159.

[7] Agarwal S, Agrawal R, Deshpande PM, Deshpande A, Gupta JF, Naughton RR, Sarawagi S. On the computation of multidimensional aggregates. In: Vijayaraman TM, ed. Proceedings of the 22nd International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1996. 506~521.

[8] Hellerstein JM, Haas PJ, Wang HJ. Online aggregation. ACM SIGMOD Record, 1997,26(3):171~182.

[9] Acharya S, Gibbons PB, Poosala V, *et al*. Join synopses for approximate query answering. ACM SIGMOD Record, 1999,28(3):275~286.

[10] Shatdal A, Naughton JF. Adaptive parallel aggregation algorithms. ACM SIGMOD Record, 1995,24(3):104~114.

[11] Yan WP, Larson P. Eager aggregation and lazy aggregation. In: Umeshwar D, ed. Proceedings of the 21st International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1995. 345~357.

[12] Graefe G. Query evaluation techniques for large databases. ACM Computing Surveys, 1993,25(2):73~130.

[13] Jiang XD, Feng JH, Zhou LZ. A new aggregation algorithm for OLAP query evaluation. Journal of Software, 2002,13(1):65~70 (in Chinese with English Abstract).

[14] Jiang XD, Zhou LZ. A multi-table join algorithm for data warehouse query processing. Journal of Software, 2001,12(2): 190~195 (in Chinese with English Abstract).

附中文参考文献:

[5] 蒋旭东,周立柱.利用实物化视图实现 OLAP 查询.兰州大学学报(自然科学版),1999,35(8):242~247.

[13] 蒋旭东,冯建华,周立柱.联机分析查询处理中的一种聚集算法.软件学报,2002,13(1):65~70.

[14] 蒋旭东,周立柱.数据仓库查询处理中的一种多表连接算法.软件学报,2001,12(2):190~195.