ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.10, October 2010, pp.2494−2512
doi: 10.3724/SP.J.1001.2010.03679
© by *Institute of Software, the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

主存 OLAP 系统中 what-if 查询处理策略*

张延松[1,2,3+], 肖艳芹[1,2,4], 王 珊[1,2], 陈 红[1,2]

[1](中国人民大学 数据工程与知识工程教育部重点实验室,北京 100872)

[2](中国人民大学 信息学院,北京 100872)

[3](中国人民大学 中国调查与数据中心,北京 100872)

[4](河北大学 计算中心,河北 保定 071002)

What-If Query Processing Policy of Main-Memory OLAP System

ZHANG Yan-Song[1,2,3+], XIAO Yan-Qin[1,2,4], WANG Shan[1,2], CHEN Hong[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

[3](National Survey Research Center at Renmin University of China, Beijing 100872, China)

[4](Computer Center, Hebei University, Baoding 071002, China)

+ Corresponding author: E-mail: zhangys_ruc@ruc.edu.cn

Zhang YS, Xiao YQ, Wang S, Chen H. What-If query processing policy of main-memory OLAP system. *Journal of Software*, 2010,21(10):2494−2512. http://www.jos.org.cn/1000-9825/3679.htm

Abstract: A what-if analysis can provide a more meaningful information than classical OLAP (on-line analysis processing). Multi-Scenario hypothesis upon historical data needs efficient what-if data view support. Two novel algorithms of deltaMap and pre-merge, which can greatly improve the performance of delta table algorithm with set operations, are proposed. To analyze the performance of query re-writing algorithm and delta cube algorithm under different what-if update conditions, a global performance analysis and comparison are presented in the experiment section. This paper proposes a cost model for a what-if analysis processing engine, based on different algorithms with parameters such as application scenario, what-if update rate, complexity of what-if updates, memory storage policy, cardinality of query result set etc, that can be used as a practical framework in a what-if analysis system.

Key words: what-if analysis; main-memory database; delta table; query re-writing; deltaMap index; pre-merge algorithm

摘 要: What-If 分析能够提供比传统的 OLAP(on-line analysis processing)分析更加有意义的决策支持信息.基于

* Supported by the National Natural Science Foundation of China under Grant Nos.60473069, 60496325 (国家自然科学基金); the National High-Tech Research and Develpoment Plan of China under Grant No.2008AA01Z120 (国家高技术研究发展计划(863)); the Joint Research of HP Lab. China and Information School of Renmin University of China (国际合作(HP 实验室)项目); the Joint Research of Beijing Municipal Commission of Education and Information School of Renmin University of China (北京市教委产学研合作项目); the Renmin University of China Graduate Science Foundation under Grant No.08XNG040 (中国人民大学研究生科学研究基金)

Received 2008-12-09; Accepted 2009-07-07

© 中国科学院软件研究所 http://www.jos.org.cn

历史数据的应用场景假设分析需要更加有效的 what-if 数据视图生成机制的支持.在传统的 delta 表合并算法的基础上,提出了基于内存记录指针的 deltaMap 算法来提高 what-if 数据视图的合并性能.根据 OLAP 分析的应用特点,提出了 pre-merge 算法来处理支持分布式计算的聚集运算.根据不同的假设更新类型,对查询重写算法和Δcube 算法作了详细的性能测试并进行了全面的性能分析对比,在此基础上提出了 what-if 分析的代价模型,以应用场景模式、假设更新率、假设更新复杂度、查询结果集的基数作为参数,有效地描述系统 what-if 查询处理策略,为 what-if 分析的解决方案提供了一个可行的框架结构.

关键词: what-if 分析;内存数据库;delta 表;查询重写;deltaMap 索引;pre-merge 算法

中图法分类号: TP311 **文献标识码**: A

What-If query 是基于假设场景的查询,what-if analysis 是基于假设场景的 OLAP(on-line analysis processing,联机分析系统)查询,在 ROLAP(relational OLAP)系统中,主要通过 *group-by* 等操作来提供基于聚集运算的查询处理,因此,what-if analysis 可以看作是 what-if query 的一个面向多维聚集计算领域的子集.支持 what-if analysis 的 OLAP 系统需要支持 ad-hoc 查询、上卷和下钻等 OLAP 应用中典型的多维分析功能.what-if analysis 的主要技术难点是,如何在数据仓库上存储和表示用户的假设更新内容以及如何在基于假设更新的数据仓库上执行高效率的 OLAP 分析查询.

在实际应用中,what-if analysis 可以作为一种有效的数据分析手段来为企业提供基于真实历史数据上的、更加灵活可靠的多维数据分析工具.what-if analysis 可以划分为如下 3 个阶段:

(1) 将假设分析的业务逻辑映射为对历史数据的假设更新;

(2) 在数据存储层或查询处理层表示假设更新;

(3) 在假设更新的基础上进行 OLAP 查询.

What-If analysis 是对基于历史数据的 OLAP 功能的扩展,不仅需要解决对假设场景分析需求的有效表示和正确处理,还要提供高性能的解决方案以避免恶化 OLAP 的性能.本文研究的重点是实现高性能的 what-if analysis 算法,并为系统提供有效的代价评估模型以实现系统决策方案的自动选择.根据已有的研究成果和应用的需求,研究的重点放在基于值的假设更新上,它是 what-if analysis 的基础,也是影响 what-if analysis 性能的关键因素.

从数据的表示方法来看,假设更新主要分为 3 种类型:

(1) 基于 delta 表.在历史数据上的假设更新被物化为 delta 记录,并保存在独立的 delta 表中.delta 表存储假设更新中更改、插入和删除的记录、相应的版本号、类型等.delta 表保持了数据仓库的数据独立性,而且可以通过多版本机制保存多用户、多版本的假设更新数据,各个版本的假设更新数据可以独立使用,能够较好地解决多用户应用场景上的假设分析问题.假设更新只需要执行一次,随后的查询可以直接在 delta 表和相应基表的合并数据视图上进行.其缺点是假设更新数据需要占用额外的存储空间,不适用于更新频率大、更新比例大、更新操作简单的应用场景.当数据量增大时,delta 表和基表的合并操作代价很大.

(2) 基于查询重写机制.当用户的假设更新可以通过查询重写来实现时,可以直接在重写的查询基础上进行 OLAP 分析操作,不需要额外的存储空间存储假设更新的数据.当遇到假设更新为简单的代数更新、假设更新影响的记录数量大、假设更新重复使用率低的应用场景时,查询重写具有很好的效率.当假设更新涉及复杂的表间连接操作、聚集运算时,查询重写的执行效率降低,而且假设更新的结果不能被重复的查询利用.当假设更新在多用户、多版本场景下产生级联假设更新时,嵌套的查询重写会极大地降低 what-if analysis 的性能.

(3) 基于Δcuboid.当系统采用物化视图机制时,可以将 delta 记录作为一个独立的数据单位生成与查询对应的Δcuboid,然后与查询对应的 cuboid 进行合并.通过 cuboid 内部的维结构信息,将Δcuboid 在 cube 中依次逐层更新以支持上卷操作.与事实表相比,cuboid 中的记录数量大为减少,因此,相应的更新操

© 中国科学院软件研究所 http://www.jos.org.cn

作代价很小,而且该算法支持基于 cube 结构的层次更新.Δcuboid 适用于采用大量物化视图的应用场景,当 cuboid 的基数很大时,cuboid 与Δcuboid 之间的合并操作性能也将大为降低.该算法是基于可分布式计算的聚集运算而设计的,具有一定的局限性,并不适用于所有的聚集操作.

我们在“基于内存的联机分析处理系统”项目的研究过程中,深入研究了各种 what-if analysis 算法,并根据不同的应用场景和数据模式设计了性能测试实验,提出了新的算法来提高 what-if analysis 的性能.为解决系统 what-if 查询策略选择问题,我们结合算法性能测试实验的结果,在本文中提出了 what-if analysis 的代价模型,为 what-if analysis 的解决方案提供一个可行的框架结构.

本文的贡献包括:

(1) 提出了新的基于 delta 表合并机制的高性能 what-if 查询算法以及基于查询重写机制的Δcuboid 合并算法,扩展了Δcuboid 算法的适应性.根据假设更新的不同类型,对查询重写算法进行性能分析和测试,并与传统的 delta 表合并算法进行对比.

(2) 设计了基于内存数据库 SQL 引擎的算法测试实验,综合测试了各种算法的性能,提供基于相同执行引擎的性能评价.

(3) 提出了 what-if 查询代价评估模型,完善了 what-if analysis 的框架体系研究内容.

本文第 1 节介绍相关的研究工作和存在的问题.第 2 节~第 4 节分别介绍 delta 表合并算法、查询重写算法和Δcuboid 合并算法的实现和扩展算法.第 5 节介绍实验设计与实验结果分析.第 6 节提出 what-if analysis 的代价模型.最后总结本文的工作并给出后续的研究方向.

1 相关工作

What-If analysis 是 OLAP 和 DSS(决策支持系统)领域内重要的分析功能,用于重要业务数据的假设场景分析.但在应用环境中,what-if analysis 主要作为一个辅助的决策分析工具对重要的企业经济参数或关键的数据视图进行假设分析,并未提供完整的基于 OLAP 模式的假设分析功能.

What-If analysis 是 what-if query 在 OLAP 领域中的应用,查询结果是基于维结构的聚集值.传统的聚集表算法[1−3]采用容易在 SQL 引擎中实现的集合并和差操作来实现 delta 表与基表的合并,根据算法的要求,将记录更新操作拆分为对原记录的删除和插入具有更新值的新记录两个操作,分别存储在删除表 T 和插入表 S 中,假设更新后的数据视图可以通过集合操作 $R\cup S-T$ 来实现.当一个记录被删除后又重新插入时,该算法将同一记录分别记录在删除表 T 和插入表 S 中,当进行集合运算时必须要通过 delta 记录附加的时间戳信息才能保证该记录的有效输出.在实际应用中,一般采用单一的 delta 表存储 *update*(*U*),*delete*(*D*),*insert*(*I*)类型的 delta 记录,并通过 delta 表中 delta 记录子集与基表之间的集合操作来生成 what-if analysis 所需的假设数据视图.传统 delta 表合并算法受低效率集合操作的制约性能较差,由于无法通过标准的 SQL 语法实现 delta 表与基表之间基于类型判断的合并操作,因此该算法的研究处于停滞状态.delta 表独立的数据存储特性和物化复杂假设更新结果的特点使其在复杂的假设更新计算、多用户、多版本的应用场景中具有较好的适应性,高性能的 delta 表合并算法研究是我们研究工作中的一个重点内容[4,5].

Sesame[6]是一个基于查询重写算法的原型系统,它通过完整的查询重写规则构建了基于 cube 结构的 what-if analysis 解决方案.查询重写算法支持所有可以基于 SQL 操作实现的假设更新操作,包括复杂假设更新所导致的表间连接操作.在查询重写过程中可以将假设删除对应的集合差操作转换为一个或多个选择操作,如 $R-\{t|t[A]>1500\wedge t\in R\}$ 可以改写为 $\delta_{A\leqslant 1500}(R)$,从而大大提高了查询执行的速度.在 3 种假设更新类型 *U*,*I*,*D* 中,*I* 类型的假设更新记录可以通过子查询表示,并通过集合并操作与基表合并,但 *U* 类型和 *D* 类型的假设更新记录需要根据假设更新的约束条件生成一系列的查询子集并进行合并.如图 1 所示,当在一个属性上设置假设区间时,假设数据视图由 3 个属性区间组成;当在两个属性上设置假设区间时,假设数据视图由 9 个数据区域(优化合并为 5 个)组成;当在 3 个属性上设置假设区间时,假设数据视图由 27 个(优化合并为 7 个)数据空间组成.当事实表中存在 n 个维字段时,假设更新对应的数据空间是一个超立方体,我们假定假设更新在每个维上分割的数据区

© 中国科学院软件研究所 http://www.jos.org.cn

域为 $m_1,m_2,\ldots,m_n$ 个,则假设数据视图将由 $\prod_{i=1}^{n} m_i$ 个数据子空间合并形成.当存在交迭的假设更新时,假设数据视图的成员数据空间数量将会更多,查询改写变得更加复杂.

另一方面,查询重写方式不能存储查询结果,每次执行 what-if analysis 查询时都需要实时地生成假设更新视图.当假设更新需要进行复杂的运算但假设更新影响的记录数量很少时,基于查询重写的 what-if analysis 查询难以提供较高的查询处理性能.

OLAP 应用中主要是针对 cube 结构的数据立方体进行基于聚集运算的查询处理,cube 由 2^n 个 cuboid 组成,在 what-if analysis 应用场景下,用户的假设更新产生相应的Δcube,如图 2 所示.当系统处理用户的 OLAP 查询时,可以将相应的 cuboid 和Δcuboid 合并,并通过合并后的假设数据视图完成查询处理.Δcube 算法能够充分利用已有的物化视图来提高 what-if analysis 的查询处理性能,由于假设更新所产生的 delta 记录相对基表记录要少得多,因此Δcuboid 的创建和Δcube 的产生只需要消耗很少的时间代价,而且Δcube 中的记录数量也远远小于 cube 中的记录数量.因此,cuboid 和Δcuboid 的合并操作也具有良好的性能.文献[7,8]就Δcube 机制的增量 cube 维护进行了研究,提出了相关的实现算法.但该算法的局限性在于,基于 cube 的增量维护只能支持可分布计算或可代数运算的聚集操作,如 *SUM,AVERAGE,COUNT* 等,对于其他不可分布计算的聚集运算,如 *MEDIAN* 等聚集运算不能应用该算法,只能重新计算 cuboid 并重构整个 cube.

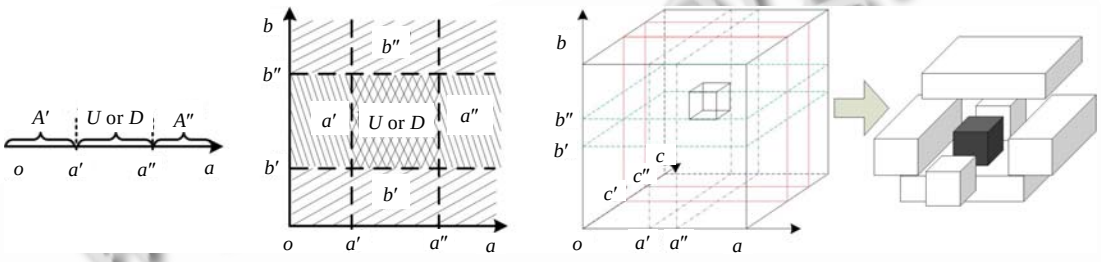


Fig.1 Query re-writing with multi-attribute what-if update

图 1 多个属性上进行假设更新时的查询重写

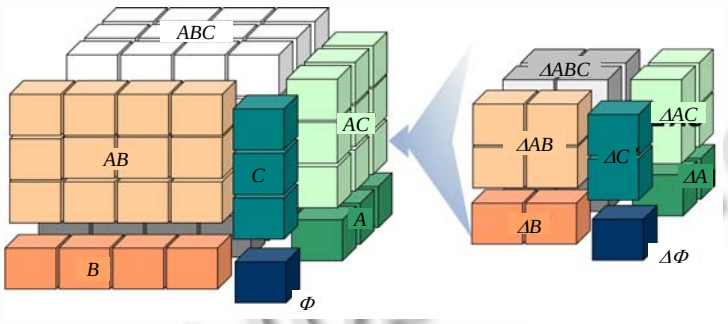


Fig.2 Cube and refresh based on Δcube

图 2 基于Δcube 的 cube 更新

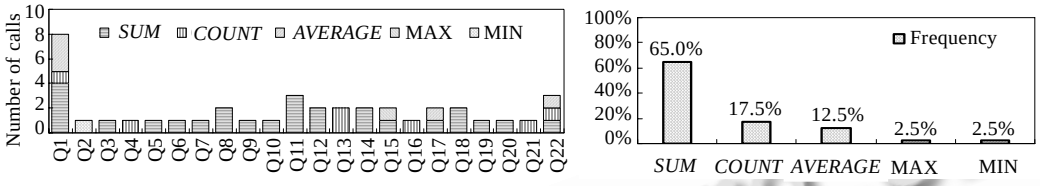
不同类型的聚集函数在 OLAP 查询中的使用频率有很大的差异,其使用频率符合 80/20 规律,即在应用环境中,绝大多数的聚集操作集中在少数几个聚集函数上.以 TPC-H 测试标准中的 22 个测试查询为例,图 3 显示了在 22 个测试查询中调用的聚集函数的次数和不同类型的聚集函数在测试查询集中的使用频率.我们可以看到,*SUM,AVERAGE,COUNT* 这 3 个聚集函数总的使用频率达到 95%.因此,基于可分布计算的聚集函数的Δcube 算法具有很好的实用性.

文献[9,10]将 OLAP 强大的查询功能与 spread sheet 灵活的计算能力结合起来,提出了 SQL spreadsheets 对

© 中国科学院软件研究所 http://www.jos.org.cn

象,从而可以利用 Excel 作为 OLAP 查询的前端处理对象.而且通过灵活的公式定义功能,可以支持很多基于 Excel 的 what-if analysis 插件,从而为 OLAP 查询增加易于集成的 what-if analysis 功能.目前,基于 spreadsheets 的解决方案还不能实现与 OLAP 服务器之间的动态交互,其 what-if analysis 功能仅限于某个特定查询视图之上的 what-if 分析,还不能提供全局的解决方案.

OLAP 查询是在海量数据的基础上进行的复杂的、带有聚集运算的查询操作,传统的基于磁盘数据库的 OLAP 难以满足用户高性能查询响应的需求,成为影响 OLAP 应用的关键制约因素.基于内存数据库的 OLAP 性能与磁盘数据库相比具有很大的优势,但在海量的数据处理过程中,性能仍然是必须要考虑的因素.what-if analysis 是对 OLAP 的扩展,能够为用户提供更灵活的决策分析信息,同时又进一步降低了 OLAP 的性能.因此,在基于 what-if analysis 的 OLAP 系统中,what-if analysis 的性能成为影响系统性能的关键因素.



(a) (b)

Fig.3 Frequency of aggregate functions in TPC-H testing queries

图 3 TPC-H 测试查询中不同函数的使用率

文献[11]中提出了 what-if 查询的框架结构,并将 delta 表合并算法和查询重写算法分别定义为 Eager,Lazy 和 Hybrid 这 3 种类型,基本的原则是:根据假设更新所影响的记录数量来决策 what-if 查询的处理方法,将数量较少的假设更新记录物化到 delta 表中,并通过合并算法支持 what-if 查询;对于假设更新影响记录数量大的简单查询,通过查询重写方法来实现 what-if 查询,减少空间的开销.当 what-if 查询包含多个假设更新场景时,如果不同的假设更新场景中所影响的记录数量有较大的差异,则可以根据不同的假设更新场景选择不同的假设更新表示方法,通过混合模式的假设更新来支持 what-if 查询.

What-If analysis 具有复杂的应用背景、不同的数据场景和不同的实现方法,目前还没有在统一平台上进行的全面的性能分析方面的研究成果,选择不同 what-if 查询处理策略的代价评估模型方面的研究也没有取得有效的进展.本文研究的重点是根据 OLAP 查询的特点扩展传统的 what-if 分析算法,并基于统一的内存数据库平台进行性能分析与实验,获得不同算法在不同数据与应用场景下的性能特点,探讨 what-if analysis 的代价评估模型,优化 what-if analysis 系统的存储策略和查询执行策略.

2 delta 表合并算法

2.1 delta表合并算法执行过程分析

我们首先定义文中的相关概念和术语.

定义 1(基表(base table)). 基表为假设更新的基础数据表,可以是数据仓库中的任意关系,也可以是某个物化的查询视图[12].在基于值假设更新的应用场景中,基表主要指的是事实表,我们记为

$$B(D_1,D_2,\ldots,D_m,M_1,M_2,\ldots,M_n),$$

其中,$D_1,\ldots,D_m$ 为维属性,$M_1,\ldots,M_n$ 为度量值属性.

定义 2(delta 表(delta table)). delta 表为在基表上进行假设更新的物化数据集,除与基表对应的属性外,还需要增加若干个附加属性来表示假设更新的类型和版本等信息.我们记为

$$D(d_1,d_2,\ldots,d_m,m_1,m_2,\ldots,m_n,VER,\_count,FLAG),$$

其中,*VER* 表示假设更新记录的版本信息,*FLAG* 表示假设更新的类型.在本文中,假设更新包括 *update*,*insert* 和

© 中国科学院软件研究所 http://www.jos.org.cn

delete 这 3 种类型,分别以 *U*,*I* 和 *D* 来表示.对于假设更新的度量属性,可以采用两种存储策略:一是存储假设更新终值,即更新后的属性值;二是存储假设更新差值,即在 *update* 更新操作中,$m_i=V_i-M_i,i\in[1,n]$,将在度量属性 M_i 上的假设更新的终值 V_i 和原度量属性值 M_i 的差值存储在 delta 记录中,未发生假设更新的度量属性值设置为 0;在 *delete* 更新操作中,$m_i=-m_i$;在 *insert* 更新操作中,$m_i=V_i$.*_count* 字段存储 delta 记录的计数信息,*U* 类型为 0,*I* 类型为 1,*D* 类型为−1.

在基于集合差运算的 delta 表合并算法中,delta 表被分成插入表和删除表,*update* 操作被分解为对基表中原始记录的删除和插入新记录两个操作,分别在删除表和插入表中创建一个 delta 记录.文献[3]提出了通过时间戳来解决 re-insertion 问题,因此我们将两个 delta 表分别定义为:

- $D_D(d_1,d_2,\ldots,d_m,m_1,m_2,\ldots,m_n,TIMESTAMP)$,度量属性中存储的是基表中原始记录的属性值;
- $D_I(d_1,d_2,\ldots,d_m,m_1,m_2,\ldots,m_n,TIMESTAMP)$,度量属性中存储的是假设更新后的终值.

文中如无特殊说明,delta 表中的度量字段采用差值存储策略.我们用符号“⟕”表示基表和 delta 表之间的合并操作,则假设更新的数据视图可以表示为

$$B⟕D=\{b|b\in B\}⟕\{d|d\in D\},$$

其结果集可以表示为

$$\{t(D_1,\ldots,D_m,M_1,\ldots,M_n)|t\in B\}\cup\{t(D_1,\ldots,D_m,M_1+m_1,\ldots,M_n+m_n)|t\in D\wedge t(FLAG)\text{ in }\{\text{‘}U\text{’},\text{‘}D\text{’}\}\cup$$
$$\{t(d_1,\ldots,d_m,m_1,\ldots,m_n)|t\in D\wedge t(FLAG)=\text{‘}I\text{’}\}.$$

结果集中的第 1 部分为基表中未被假设更新影响的记录,即 *B*-*D* 的结果;第 2 部分为基表中被假设更新影响的记录集,其度量属性值为基表与 delta 表中对应记录度量值的和;第 3 部分为基表中不存在的、在假设更新时所增加的新记录.由结果集的内容我们可以看到,基表与 delta 表合并的计算代价主要在于基表与 delta 表之间的集合差运算代价和基表与 delta 表之间相对应的 delta 记录之间连接和重计算度量值的代价.根据对 MonetDB,PostgreSQL 和 SQL server 等查询引擎的测试,集合差操作的执行代价远大于连接操作.因此,从算法的执行层次来看,算法优化的一个可行方案是通过连接操作替代集合差操作.

定义 3(分析查询(analysis query)). 将基于 cuboid 的查询定义为分析查询,这是指基于指定维属性的分组聚集操作,记为

$$\psi_{O,A}\{D_1,\ldots,D_m,M_1,\ldots,M_n\},$$

其中,$O=\{SUM,COUNT,AVG,\ldots\}$,*A* 为聚集操作所对应的度量属性集合.

我们以维属性 D_1 进行分组,并在度量属性上以进行 *SUM* 聚集操作为例给出分析查询的表达式.

$$\psi_{O=SUM,A=\text{‘}D_1\text{’}}\{D_1,\ldots,D_m,M_1,\ldots,M_n\}=\bigcup_{j=1}^{k}\{d_j,\sum_{i=1}^{card(\delta_{D_1=\text{‘}d_j\text{’}}(B))}M_1,\ldots,\sum_{i=1}^{card(\delta_{D_1=\text{‘}d_j\text{’}}(B))}M_n\}\,|\,d_1,\ldots,d_k\in D_1.$$

我们通过 $B⟕D$ 结果集的不同子集,考察聚集操作 *SUM* 在分析查询中的特性.

$$\sum_{i=1}^{card(\delta_{D_1=\text{‘}d_j\text{’}}(B))}M_k=\sum\{t(D_1,\ldots,D_m,M_1,\ldots,M_n)\,|\,t\in B\}+\sum\{t(D_1,\ldots,D_m,M_1+m_1,\ldots,M_n+m_n)\}+$$
$$\sum\{t(d_1,\ldots,d_m,m_1,\ldots,m_n)\}=\sum\{t\,|\,t\in B\}+\sum\{t\,|\,t\in D\}=\sum\{t\,|\,t\in B\cup D\}.$$

即,在 *SUM* 聚集运算中,记录集满足结合律.因此,基于 *SUM* 运算的分析查询支持基于基表和 delta 表预合并机制的分组聚集操作,从而在 what-if analysis 查询中节省基表与 delta 表合并的运算代价.

对于聚集操作 *COUNT* 而言,基表记录可能对应一个 *delete* 类型的 delta 记录,或多个 *update* 类型的 delta 记录.因此,直接采用预合并机制进行分析查询运算会产生错误的结果.对此,我们设计了如下预合并机制:

(1) 在预合并时,基表产生一个临时字段*_count*,初值设置为 1;

(2) 在预合并时,delta 表中的字段*_count* 存储记录计数信息,*U*(*update*)类型的 delta 记录的*_count* 字段设置为 0,*I*(*insert*)类型的 delta 记录设置为 1,*D*(*delete*)类型的 delta 记录设置为−1;

(3) 在一个假设更新版本中,同一个基表记录可能对应多个 *U* 类型的 delta 记录,其*_count* 值均设为 0.

在预合并时,我们以 *SUM*(*count*)的结果来代替 *COUNT*(*)的结果,对于不同的 delta 记录类型,结果分别为:

- *U*: 1(*B._count*)+0(*D._count*)+0(*D._count*)+…=1;

© 中国科学院软件研究所 http://www.jos.org.cn

- *I*: 1(*D._count*)=1;
- *D*: 1(*B._count*)+(−1)(*D._count*)=0.

在假设更新时,一个基表记录可以产生多个假设更新的 delta 记录,而同一基表记录上只能进行一次假设删除.假设删除后,基表记录的 *re-insert* 操作产生一个新的 *I* 类型的假设插入记录,最终的记录计数为 1+(−1)+1=1,不影响 *COUNT* 操作结果的正确性.因此,该预处理合并方法也适用于多版本的应用场景.

AVERAGE 运算的结果可以用 *SUM*/*COUNT* 来产生,因此,预合并机制可以支持高使用率的聚集运算 *SUM*, *COUNT* 和 *AVERAGE*,可以应用在大多数的分析查询应用场景.

2.2 delta表合并算法扩展研究

2.2.1 基于集合操作的 delta 表合并算法

算法采用关系代数描述如下:

算法 1. setBDM (set operation based delta table merging algorithm).

输入:基表 $\boldsymbol{B}$、delta 表 $\boldsymbol{D_D}$ 和 $\boldsymbol{D_I}$;

输出:假设更新数据视图 what-ifView.

$$what\text{-}ifView=(\boldsymbol{B}-\boldsymbol{D_D}\cup\boldsymbol{D_I}).$$

实际应用中,一般采用统一的 delta 表来存储假设更新的记录.扩展的基于集合合并操作的算法描述为:

算法 2. joinBDM (join operation based delta table merging algorithm).

输入:基表 $\boldsymbol{B}$、delta 表 $\boldsymbol{D}$;

输出:假设更新数据视图 what-ifView.

$$what\text{-}ifView=(B\bowtie(\pi_{(d_1,\ldots,d_m)}(B)-\pi_{(d_1,\ldots,d_m)}\delta_{FLAG='D'}(D)))\cup(\pi_{(d_1,\ldots,d_m,m_1,\ldots,m_n)}\delta_{FLAG='U'\text{ or }FLAG='I'}(D))$$

在 joinBDM 算法中,首先排除出基表中被假设 *delete* 影响的记录,然后再与假设更新后出现在假设更新数据视图中的 *update* 和 *insert* 类型的 delta 记录合并.

与算法 1 相比,算法 2 中只需要维护一个 delta 表,并简化了假设更新操作,不需要将 *U* 类型的假设更新操作分解为 *D* 和 *I* 两个操作.但从算法执行的时间复杂度上看,两种算法都需要进行差运算操作,其时间复杂度接近.

2.2.2 基于基表记录与 delta 表记录映射关系的 delta 表合并算法

我们从基表记录与 delta 表记录之间的映射关系的角度来分析 delta 表合并算法,如图 4 所示,基表记录与 delta 表记录之间存在 4 种类型的映射关系:

1) 基表记录没有对应的 delta 记录,如图中第 2 条、第 5 条记录所示;
2) 基表记录在 delta 表中存在 1 条 *U* 类型的 delta 记录,如图中第 1 条记录所示;
3) 基表记录在 delta 表中存在 1 条 *D* 类型的 delta 记录,如图中第 3 条记录所示;
4) delta 表中存在 1 条 *I* 类型的 delta 记录,如图中第 4 条记录所示.

Base table						Delta table							
D_1	D_2	D_3	M_1	M_2	M_3	d_1	d_2	d_3	m_1	m_2	m_3	_count	FLAG
1	2	1	3.5	5.6	7	1	2	1	0	0	2	0	*U*
2	3	7	5.8	7.6	9								
3	2	4	5.1	2.8	6	3	2	4	−5.1	−2.8	−6	−1	*D*
						4	2	6	1.4	5.7	3	1	*I*
4	5	9	7.4	5.1	9								

Fig.4 Mapping relation of base tuples and delta tuples

图 4 基表和 delta 表记录之间的映射关系

在映射表的基础上可以根据基表记录和delta记录的内容确定当前输出的记录.在SQL引擎中,基表与delta表之间的映射表可以通过 *B* full join *D* 来获得,算法描述为:

算法 3. fullJoinBDM (full join based delta table merging algorithm).

输入:基表 $\boldsymbol{B}$、delta 表 $\boldsymbol{D}$;

© 中国科学院软件研究所 http://www.jos.org.cn

输出:假设更新数据视图 what-ifView.

tmpView=***B*** full join ***D***;

for each tuple *t* in *tmpView*

if $t.D_1$ IS NOT *NULL* AND *t.FLAG* IS *NULL* **then** *output*($t.D_1$,…,$t.D_m$,$t.M_1$,…,$t.M_n$)→what-ifView;

if $t.D_1$ IS NOT *NULL* AND *t.FLAG*='*U*' **then** *output*($t.D_1$,…,$t.D_m$,$t.M_1+t.m_1$,…,$t.M_n+t.m_n$)→what-ifView;

if $t.D_1$ IS NOT *NULL* AND *t.FLAG*='*D*' **then** skip *t*;

if $t.D_1$ IS *NULL* AND *t.FLAG*='*I*' **then** *output*($t.d_1$,…,$t.d_m$,$t.m_1$,…,$t.m_n$)→what-ifView;

end each

return what-ifView;

算法 3 中,临时表的空间开销是(|*B*|+$ratio_I$×|*D*|)×(*width*(*B*)+*width*(*D*)).由于 *I* 类型的假设更新记录在基表中无映射对象,因此可将 *I* 类型的 delta 记录与结果集直接合并,从而节省 full join 连接的时间代价,并简化了算法中的判断条件,算法描述如下:

算法 4. leftJoinBDM (full join based delta table merging algorithm).

输入:基表 ***B***、delta 表 ***D***;

输出:假设更新数据视图 what-ifView.

tmpView=***B*** left join ***D***;

for each tuple *t* in *tmpView*

if *t.FLAG* IS *NULL* **then** *output*($t.D_1$,…,$t.D_m$,$t.M_1$,…,$t.M_n$)→what-ifView;

if *t.FLAG*='*U*' **then** *output*($t.D_1$,…,$t.D_m$,$t.M_1+t.m_1$,…,$t.M_n+t.m_n$)→what-ifView;

if *t.FLAG*='*D*' **then** skip *t*;

end each

return $what\text{-}ifView \cup \pi_{(d_1,\ldots,d_m,m_1,\ldots,m_n)}\delta_{FLAG='I'}(D)$;

算法 4 中,临时表的空间开销是|*B*|×(*width*(*B*)+*width*(*D*)).在前面介绍的两种基于连接操作的 delta 表合并算法中,每次访问 what-if 数据视图时都要生成临时的连接表.what-if analysis 工作在多用户模式下,在 what-if 数据视图上可能进行多个切片、切块、旋转等 OLAP 操作,每次操作时临时表重复生成会造成极大的 CPU 资源浪费,而且存储该临时表又增加了系统的空间开销.在此基础上,我们提出了一种新的索引结构 deltaMap,用于存储基表与 delta 表之间的映射关系.

定义 4(deltaMap). deltaMap 是一种应用于 what-if analysis 场景下的 join index[13]索引,它只记录基表记录与 delta 表记录映射中对应记录的位置信息.在内存数据库中,该位置信息可以是对应记录的物理地址或相对于起始物理地址的偏移地址;在磁盘数据库中为记录位置,表示为[块号,块内偏移地址].在本文中,我们主要研究基于内存数据库平台的 deltaMap.deltaMap 表示为如下形式:

$$deltaMap=\{(bptr_i,dptr_i)|i\in N\wedge bptr_i\rightarrow t\in B\wedge dptr_i\rightarrow t\in D\}.$$

deltaMap 可以看作是如图 4 所示的 mapping relation 的一种基于内存地址的紧凑格式表示方法,它在第 1 次生成 what-if 数据视图时同步创建,并在其他 what-if 数据视图访问时被重复使用.当数据仓库进行周期性的增量更新时,可以增量地更新 deltaMap,也可删除当前 deltaMap,并在更新后第 1 次访问 what-if 数据视图时重新创建.

基于 deltaMap 的 what-if 数据视图访问算法描述如下:

算法 5. deltaMap merging algorithm.

输入:基表 ***B***、delta 表 ***D***、索引 deltaMap;

输出:假设更新数据视图 what-ifView.

for each tuple t in deltaMap

if *t.bptr* IS NOT *NULL* AND *t.dptr* IS *NULL* **then**

© 中国科学院软件研究所 http://www.jos.org.cn

output($t.bptr\rightarrow D_1,\ldots,t.bptr\rightarrow D_m,t.bptr\rightarrow M_1,\ldots,t.bptr\rightarrow M_n$)→what-ifView

if *t. bptr* IS NOT *NULL* AND $t.dptr\rightarrow FLAG$='*U*' **then**

output($t.bptr\rightarrow D_1,\ldots,t.bptr\rightarrow D_m,t.bptr\rightarrow M_1+t.dptr\rightarrow m_1,\ldots,t.bptr\rightarrow M_n+t.dptr\rightarrow m_n$)→what-ifView

if *t. bptr* IS NOT *NULL* AND $t.dptr\rightarrow FLAG$='*D*' **then** skip *t*;

if *t. bptr* IS *NULL* AND $t.dptr\rightarrow FLAG$='*I*' **then**

output($t.dptr\rightarrow d_1,\ldots,t.dptr\rightarrow d_m,t.dptr\rightarrow m_1,\ldots,t.dptr\rightarrow m_n$)→what-ifView

end each

return what-ifView;

2.2.3 基于预合并机制的 delta 表合并算法

如第 2.1 节中所述,当分析查询的聚集操作类型为 *SUM*,*COUNT* 和 *AVERAGE* 时,根据聚集运算结合律,我们可以将基表与 delta 表合并后直接进行分组聚集运算.算法描述如下:

算法 6. pre-mergeBDM (pre-merge based delta table merging algorithm).

输入:基表 ***B***、delta 表 ***D***;

输出:假设更新数据视图 what-ifView.

if *aggregate operation*='*SUM*' **then**

{

$tmpView = B \cup \pi_{(D_1,\ldots,D_m,M_1,\ldots,M_n)}(D)$;

what-ifView=getSUMGroup-byResults(*tmpView*($M_1,\ldots,M_n$));

}

if *aggregate operation*='*COUNT*' **then**

{

$tmpView = \pi_{(D_1,\ldots,D_m,M_1,\ldots,M_n,1)}(B) \cup \pi_{(D_1,\ldots,D_m,M_1,\ldots,M_n,\_count)}(D)$;

what-ifView=getSUMGroup-byResults(*tmpView*(*_count*));

}

if *aggregate operation*='*AVERAGE*' **then**

{

$tmpView = \pi_{(D_1,\ldots,D_m,M_1,\ldots,M_n,1)}(B) \cup \pi_{(D_1,\ldots,D_m,M_1,\ldots,M_n,\_count)}(D)$;

what-ifView=getSUMGroup-byResults(*tmpView*(M_1)/*tmpView*(*_count*),…,*tmpView*(M_n)/

tmpView(*_count*));

}

return what-ifView;

在本节中,我们在经典的基于集合操作的 delta 表合并算法 1 的基础上,基于 SQL 平台进行扩展并提出了改进的 delta 表合并算法.其中,算法 3 和算法 4 基于基表与 delta 表的内在数据联系并结合 SQL 引擎的功能实现;算法 5 利用内存 OLAP 服务器平台的内存数据直接访问模式来建立基表记录与 delta 记录之间的直接访问,能够更有效地发挥内存 OLAP 服务器的优势;算法 6 虽然在应用上具有一定的局限性,但充分利用 OLAP 查询聚集运算的特点,简化 what-if 查询的数据视图合并策略,提供更高的查询处理性能.

3 查询重写算法

在查询重写算法中,不同的假设更新类型所对应的假设数据视图其生成代价各不相同.对于 *I* 类型的假设更新 delta 记录,其代价是 delta 记录与基表的 *UNION* 操作代价,如果事先已进行过假设插入记录的实体完整性约束检查,则可以使用 *UNION ALL* 操作,其执行代价很小.对于 *D* 类型的 delta 记录,假设数据视图生成代价是在基表中去掉 *D* 类型 delta 记录对应的基表记录,如果使用集合差运算,则在 SQL 引擎中的执行性能很差,提高效

© 中国科学院软件研究所 http://www.jos.org.cn

率的方法是将集合差操作转换为多维数据空间的合并操作.当数据仓库中的维数为 n 时,一个多维数据立方体的假设删除操作需要转换为 $\prod_{i=1}^{n} m_i - 1$ 个数据子空间的合并操作(m_i 为假设更新在第 i 个维上分割的数据区域个数).U 类型的 delta 记录是在 D 类型 delta 记录的处理过程的基础上增加 U 类型 delta 记录的合并过程,我们分别将两种操作需要合并的子空间数量记为 N_D 和 N_U.

$$dataSpaceWithoutDeleteTuples = \bigcup_{i=1}^{N_D} subDataSpaceOf\ (B)_i,$$

$$dataSpaceWithUpdateTuples = \bigcup_{i=1}^{N_D} subDataSpaceOf\ (B)_i \cup QueryresultsOfUpdate(B).$$

在 what-if analysis 的应用场景中,假设分析可能会基于多种应用场景进行多阶段的分析.在 delta 表方法中,可以将同一个假设更新版本中对同一个基表记录的多次假设更新的最终结果记录为一个 delta 记录,即多次的假设更新操作可以分阶段进行,delta 表中只存储最终结果.而在查询重写模式下,中间结果无法被记录,分阶段的假设更新只能转为多个嵌套的 SQL 命令,如果没有针对 what-if analysis 设计的专用查询重写优化器而单纯依赖 SQL 引擎的优化器,则其重写后的查询在执行时很难得到有效的优化.文献[11,12]中分别介绍了基于查询重写的原型系统和优化模型,但考虑到应用需求的复杂性,实现一个通用的查询重写优化引擎具有很大的难度.

本文以简单查询为例来描述查询重写算法,并在 SQL 平台上分别实现基于简单查询重写和基于复杂查询重写模式的算法,并通过实验测试和分析评估查询重写算法的性能特点.当假设更新操作 *update*,*delete* 和 *insert* 所影响的记录没有重叠时,算法描述如下:

算法 7. re-writingQuery algorithm without overlap.

输入:基表 ***B***;

输出:假设更新数据视图 what-ifView.

tmpInsertTuples=*QueryResultsOfInsert*(***B***);

tmpDeleteTuples=*QueryResultsOfDelete*(***B***);

tmpUpdateTuples=*QueryResultsOfUpdate*(***B***);

what-ifView=***B***–(*tmpDeleteTuples*∪*tmpUpdateTuples*)∪*tmpInsertTuples*∪*tmpUpdateTuples*;

return what-ifView;

当不同类型的假设更新操作所影响的记录之间存在交叠时,在查询重写时需要根据假设更新的时间顺序对 *dataView* 进行嵌套查询重写.

$$what\text{-}ifViewOfInsert = dataView \cup QueryresultsOfInsert(dataView),$$

$$what\text{-}ifViewOfDelete = \bigcup_{i=1}^{N_D} subDataSpaceOf\ (dataView)_i,$$

$$what\text{-}ifViewOfUpdate = \bigcup_{i=1}^{N_D} subDataSpaceOf\ (dataView)_i \cup QueryresultsOfUpdate(dataView).$$

4 Δcube 合并算法

文献[7,8]讨论了基于Δcube 的合并算法,并分析了各种类型的聚集运算:*SUM*,*COUNT* 和 *AVERAGE* 可以直接被算法支持;而 MAX 和 MIN 聚集函数需要扩展算法,而且只能在一定的数据场景下被支持;*MEDIAN* 等其他聚集函数不能被算法支持,只能采取重新生成 cuboid 的策略.由于对 MAX 和 MIN 函数的支持受记录数值分布的影响,因此没有完全的解决方案,在不符合算法要求时只能采取重新计算的模式.因此,在本文的Δcube 合并算法中不考虑 MAX 和 MIN 函数.

实现算法参见文献[7],本文给出在不同数据场景下生成查询所需的Δcuboid 的应用算法,具体算法描述如下:

© 中国科学院软件研究所 http://www.jos.org.cn

算法 8. Δcuboid merging algorithm based on delta table.

输入:cuboid,delta 表 ***D***;

输出:假设更新数据视图 what-ifView.

if *EXIST*(*cuboid*) **then**

{

Δcuboid=generateCuboid(***D***);

what-ifView=cuboid⋈*Δcuboid*; //⋈代表文献[7]中的Δcuboid 合并算法;

}

else

{

Δcuboid=generateCuboid(***D***);

cuboid=generateCuboid(***B***);

what-ifView=cuboid⋈*Δcuboid*;

}

return what-ifView;

算法 8 采用基于 delta 表存储策略的Δcube 合并算法,Δcuboid 合并操作的时间复杂度与连接操作相同.由于|*cuboid*|<<|***B***|,因此合并操作的性能很高.即使在没有物化视图时,分别生成 cuboid 和Δcuboid 后再进行合并,其时间代价也常常小于 delta 合并算法,其性能的差异主要决定于|*cuboid*|的大小.因此,当判断采用何种算法时,可以以假设更新数据视图数据集的基数作为算法选择的判断阈值.

当用户的假设更新为简单的代数更新时,我们结合查询重写算法和Δcube 合并算法的特点,提出了基于查询重写模式的Δcube 合并算法,通过查询重写策略表示 delta 记录集,具体算法描述如下:

算法 9. Δcuboid merging algorithm based on re-writing query.

输入:cuboid;

输出:假设更新数据视图 what-ifView.

tmpInsertTuples=QueryResultsOfInsert(***B***);

tmpDeleteTuples=QueryResultsOfDelete(***B***);

tmpUpdateTuples=QueryResultsOfUpdate(***B***);

rwDeltaSet=tmpInsertTuples∪*tmpDeleteTuples*∪*tmpUpdateTuples*;

if *EXIST*(*cuboid*) **then**

{

Δcuboid=generateCuboid(*rwDeltaSet*);

what-ifView=cuboid⋈*Δcuboid*;

}

else

{

Δcuboid=generateCuboid(*rwDeltaSet*);

cuboid=generateCuboid(***B***);

what-ifView=cuboid⋈*Δcuboid*;

}

return what-ifView;

当假设更新为影响记录多、运算简单的更新类型时,查询改写模式可以节省大量的 delta 记录存储的空间开销,但与此同时,Δcuboid 生成的代价也加大.

© 中国科学院软件研究所 http://www.jos.org.cn

综上所述,what-if analysis 查询的性能受很多因素的影响,主要的影响因素包括 what-if 更新类型、what-if 更新的影响比例、应用场景、假设更新的数据重叠比率、假设更新的复杂度、基表记录数量、what-if 数据视图的基数等.在当前的研究成果中,还没有各种算法在相同平台上的性能对比与分析,我们在 SQL 引擎中实现了各种算法并在内存数据 MonetDB 上进行了性能实验.

5 实验与性能分析

本节介绍实验的设计与配置.实验的硬件环境是:HP Integrity rx2620,2 个 1.6GHz 的 CPU,4GB 内存,160GB 硬盘.测试数据集为 FoodMart,我们设计了测试数据生成工具,测试中的事实表记录量为 800 万条事实记录,delta 记录占事实表记录的比率 η从 1%~10%,其中,*I* 类型的 delta 记录为 30%,*D* 类型的 delta 记录为 30%,*U* 类型的 delta 记录为 40%.测试查询为:

```
SELECT    store.store_state, time_by_day.the_year,
          time_by_day.quarter, product_class.product_family, SUM(sales_fact_1997.store_sales)
FROM      store,sales_fact_1997, time_by_day, product_class, product
WHERE     sales_fact_1997.store_id=store.store_id AND sales_fact_1997.time_id=time_by_day.time_id
          AND sales_fact_1997.product_id=product.product_id
          AND product.product_class_id=product_class.product_class_id
GROUP BY  store.store_state, time_by_day.the_year, time_by_day.quarter, product_class.product_family;
```

5.1 Delta表合并算法性能测试

我们根据第 2 节描述的算法综合对比了各类 delta 表合并算法,我们用 delta rate 表示假设更新数据所占的比例,图 5(a)为 *delta rate*=1%时的性能指标,图 5(b)为 *delta rate*=10%时的性能指标.其中,baseMDX 是没有假设更新时的查询执行时间,用作测试基准性能指标.

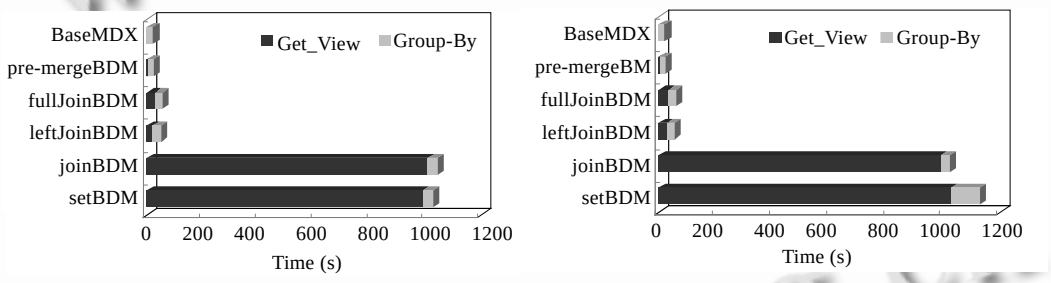


Fig.5 Analysis of performance for delta table based algorithms

图 5 基于 delta 表的合并算法性能分析

从测试结果我们可以看到,基于集合差操作的delta表合并算法性能很差,其中,数据视图合并的时间代价所占的比例超过 90%;基于连接的 delta 表合并算法的性能有很大的提高,其数据视图合并的时间代价所占的比例大约为 50%;deltaMap 算法中数据视图的合并代价大概为 30%,主要的时间代价是对各字段的判断输出与度量值合并;而基于预合并机制的 pre-merge 算法具有极低的数据视图合并代价,what-if 查询的执行性能最好.

MonetDB 是一种基于列存储模式的内存数据库,其 TPC-H 测试指标远远高于磁盘数据库和其他基于行存储模式的内存数据库.但由于其列存储模式,集合运算需要在多个属性上分别进行,其执行效率低于行存储模式.通过对其他数据库,如 PostgreSQL 和 SQL server 的测试,我们的结论是,集合差操作的性能远低于相同数据量情况下的连接操作.因此,测试的结论能够反映算法的性能特征.

如图 6 所示,我们对基于 delta 表的合并算法进行分组比较.图 6(a)为基于集合运算的合并算法,当 delta 记

© 中国科学院软件研究所 http://www.jos.org.cn

录增加时,setBDM 合并算法的执行代价随之增加;而 joinBDM 合并算法随着基表中未被假设更新影响的记录数量的减少缓慢缩短了查询执行时间.图 6(b)为基于映射关系的连接合并算法,两种算法的差异在于,leftJoinBDM 将假设插入的 delta 记录独立地与基表合并,减少了连接操作的记录量,提高了连接操作的性能.从图 6(c)中可以看到,预合并算法 pre-mergeBDM 的执行时间与原始查询的执行时间相比最多相差 15%(delta 记录比率为 10%时),deltaMap 算法需要进行额外的、时间复杂度为 $O(N+30\%\times\eta N)$的 deltaMap 索引扫描和基表记录与 delta 记录读取后合并代价,因此执行时间略多于 pre-mergeBDM 算法.但从算法的适应性方面来看,pre-mergeBDM 算法只能支持 *SUM*,*COUNT* 和 *AVERAGE* 运算,而 deltaMap 算法的通用性较强,支持各种聚集运算.图 6(d)为各种算法的综合性能比较,我们可以看到,基于集合运算的 delta 表合并算法的性能远远低于其他算法.图 6(e)对比了性能接近的几种算法,以 baseMDX 查询的执行时间为基准,4 种算法的执行时间比率为

fullJoinBDM:leftJoinBDM:deltaMap:pre-mergeBDM=2.68:2.44:1.61:1.14.

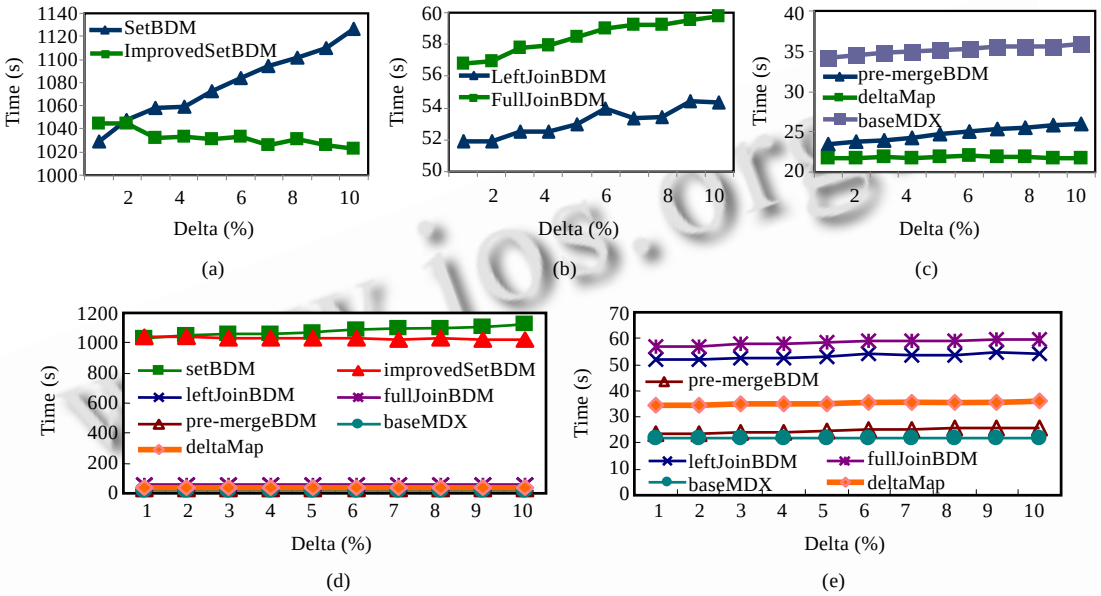


Fig.6 Performance of delta table based algorithms

图 6 基于 delta 表的合并算法性能对比

5.2 查询重写算法性能测试

在查询重写算法的性能测试实验中,我们设计了两组测试实验,分别测试简单假设更新和复杂假设更新时的算法性能.假设更新比率从 1%~10%,通过一个维上的数据选择率来控制 delta 记录比率.简单更新为代数更新,将假设更新的记录的度量值固定地提高一定的比率,复杂更新中调用了带有聚集函数的连接操作,根据连接后符合条件记录的聚集值来确定假设更新的度量值.在测试时,我们分别按照 *update*,*delete*,*insert* 和混合类型进行测试,并与相应的 delta 表合并算法进行对比.

由图 7(a)~图 7(c)可以看到,在当前设计的测试查询中,对于 *update* 和 *delete* 类型的假设更新,基于查询重写的方法是将原来的数据空间分割为多个子空间.当执行假设 *delete* 操作时合并有效的数据空间,当执行假设 *update* 时,将有效的数据子空间和更新后的数据子空间合并.与基于连接操作的 delta 表合并算法相比,查询执行性能较高.当执行假设 *insert* 更新时,查询重写算法需要生成假设 *insert* 数据子集后再与基表合并,而 delta 表合并算法直接执行基表与 delta 记录的合并操作.delta 记录相当于物化后的假设 *insert* 结果集,不必像查询重写算法那样每次重新生成 *insert* 记录集.因此,delta 表合并算法对于假设 *insert* 操作有更高的执行性能.

图 7(d)中使用混合类型的 delta 记录集,并综合比较了使用简单重写规则和复杂重写规则的查询重写算法,

© 中国科学院软件研究所 http://www.jos.org.cn

并与性能较好的 delta 表合并算法 leftJoinBDM,deltaMap 和 pre-merge 算法进行对比.测试结果表明,查询重写算法的性能低于 deltaMap 和 pre-merge 算法,简单查询重写算法的性能略高于基于连接代价的 leftJoinBDM 算法,但复杂的查询重写算法的性能低于 leftJoinBDM 算法.如果查询重写是基于多维数据上的假设更新,并且需要和多表进行连接操作,则算法的性能将进一步恶化.

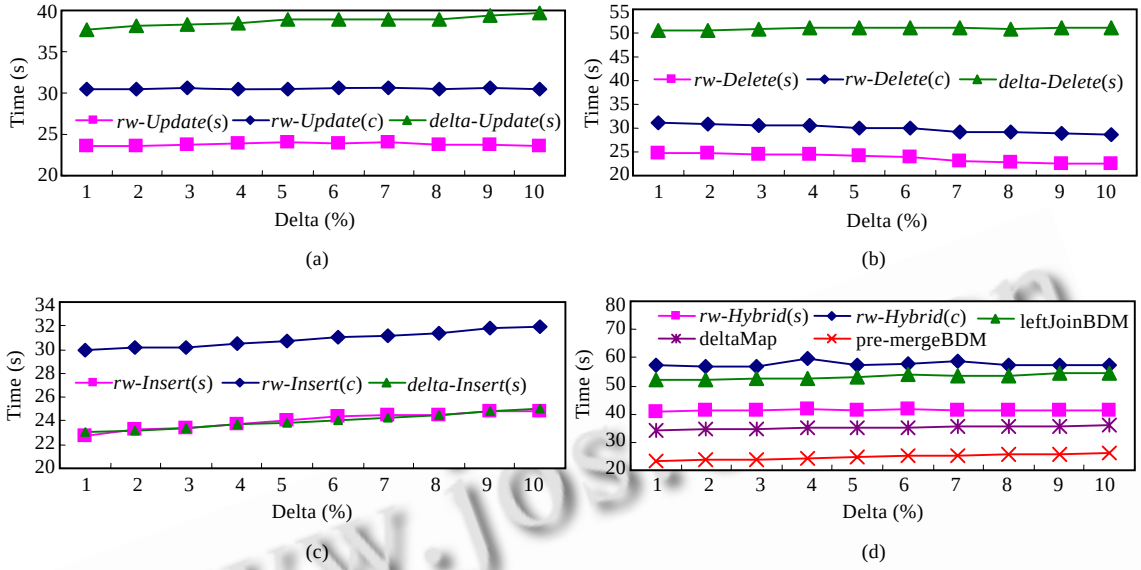


Fig.7 Performance of re-writing query algorithms

图 7 查询重写算法性能对比

5.3 Δcube合并算法

内存数据库具有较高的查询处理性能和相对有限的内存容量,在内存 OLAP 系统中,一般不存储物化视图或只存储少量使用频度较高的物化视图.因此,我们主要测试基于Δcuboid 的合并算法,以解决 ad-hoc 查询为主,cube 的维护策略与磁盘数据库中的 cube 维护策略相同.

如图 8 所示,我们测试了基于 delta 表和基于查询重写模式的Δcube 合并算法,并分别测试了基于物化视图合并和无物化视图时实时计算 cuboid 和Δcuboid 后再进行合并两种模式的查询性能.查询结果包括 4 个维属性和一个度量值属性,产生 120 个查询记录.实验结果表明:当查询结果集的基数远小于事实表时(|*cuboid*|<<|***B***|),无论是利用物化视图还是即时生成物化视图后再与 delta 记录生成的Δcuboid 进行合并,都能获得较好的执行性能.但在实验中,基于查询重写生成Δcuboid 后再与 cuboid 合并的算法的性能并不理想,原因分析如下:

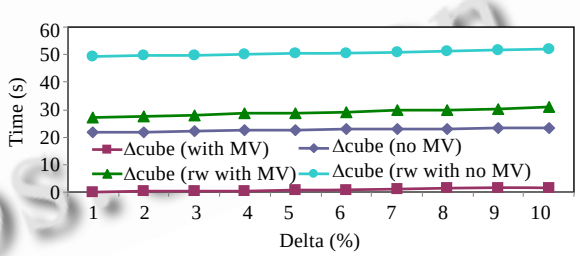


Fig.8 Performance of Δcube algorithms

图 8 Δcube 合并算法性能对比

- 基于 delta 表生成Δcuboid 时,进行 *group-by* 操作的记录数量为|***D***|,1%×|***B***|≤|***D***|≤10%×|***B***|;
- 基于查询重写生成Δcuboid 时,需要在基表上分别通过假设更新条件生成 *update*,*delete* 和 *insert* 类型的 delta 记录子集并合并,需要对事实表进行多次扫描操作,然后再进行 *group-by* 操作,增加了记录预处理的代价.测试表明,在Δcuboid 的生成过程中,基于查询重写的假设更新的时间代价超过 80%;
- MonetDB 是一种基于列存储模式的内存数据库系统,表中的各数据列分别存储为独立的数据结构.因此,在独立数据列上的数据处理性能非常高.但对于基于行扫描的操作,如 *table scan*,*union*,*except* 等操

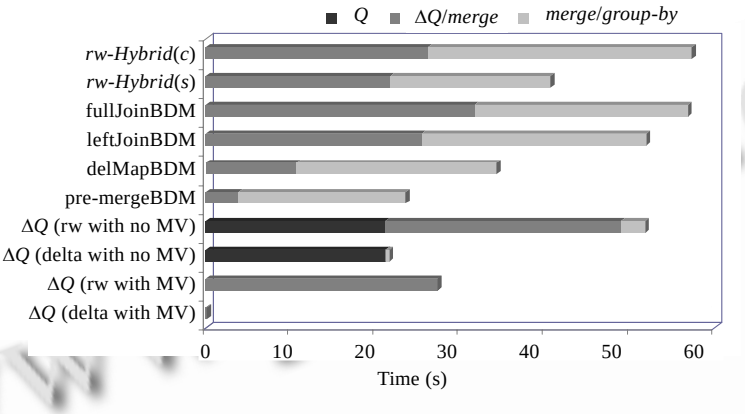
© 中国科学院软件研究所 http://www.jos.org.cn

作需要分别在数据列上进行数据处理,然后再将各列操作的结果进行连接合并.因此,对于需要根据多个维字段值进行集合操作判断的 *union* 操作,MonetDB 不能提供很好的性能,因此影响了查询重写算法的整体性能;

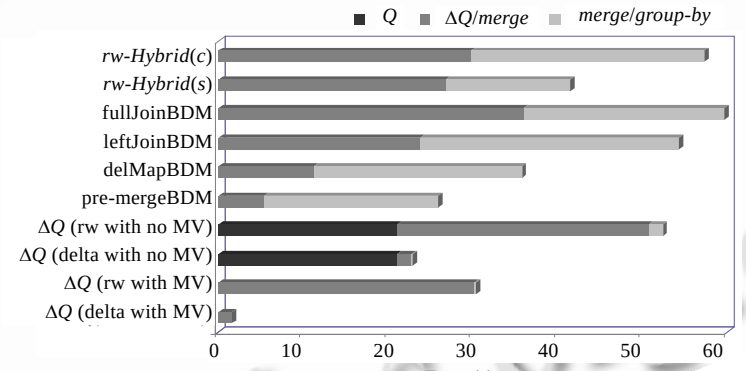
- 复杂的查询重写需要在每次调用时通过多表连接操作或聚集运算来生成假设更新的记录集,其复杂度随记录量的增大而有所增加.

5.4 What-If analysis算法性能综合分析

本文研究的 what-if analysis 查询算法包括 3 种类型:基于 delta 表合并、基于查询重写和基于Δcuboid 合并,图 9 显示了 3 类算法、9 种实现算法的具体执行性能.图 9(a)中的 delta 记录比率为 1%,图 9(b)中的 delta 记录比率为 10%.前两类算法执行过程包括 what-if 数据视图的合并和基于该数据视图的分析查询执行过程两个阶段, Δcuboid 合并算法包括 cuboid(简写为 *Q*)创建、Δcuboid(简写为Δ*Q*)创建和 cuboid 与Δcuboid 合并 3 个过程.



(a)



(b)

Fig.9 Performance of Δcube algorithms

图 9 Δcube 合并算法性能对比

根据测试结果我们可以得出如下结论:

(1) 在Δcuboid 算法中,cuboid 的记录数量固定,因此合并的代价基本恒定,主要的执行代价为 cuboid 和 Δcuboid 的生成代价.当存在物化视图时,算法的执行性能很高;

(2) 基于查询重写模式的Δcuboid 生成过程的执行代价远远高于基于 delta 表的Δcuboid 生成代价;

(3) 在基于 delta 表的算法中,数据视图的生成代价占了很大的比例,pre-merge 算法具有较高的执行性能但只支持有限的聚集操作,leftJoinBDM 算法可以看作是 fullJoinBDM 算法的优化执行策略,deltaMap

© 中国科学院软件研究所 http://www.jos.org.cn

算法可以看作是 fullJoinBDM 算法的扩展,其执行性能相当于对 full join 物化视图的访问性能,deltaMap 索引的存储空间开销与 full join 物化视图的空间开销的比值为(*width*(*A*)表示字段 *A* 的宽度)

$$(width(bptr)+width(dptr))/(2\times(\sum_{i=1}^{m}width(D_i)+\sum_{j=1}^{n}width(M_j))+width(VER)+width(\_count)+width(FLAG)).$$

(4) 当假设更新为简单的代数更新且假设更新所产生的数据子空间较少时,基于查询重写模式的算法的性能优于基于 delta 表的合并算法;当假设更新为复杂更新时,基于查询重写模式的算法的性能低于基于 delta 表的合并算法;

(5) 当假设更新的数据视图需要复杂的计算并且被多次访问时,基于查询重写的算法需要进行大量复杂的重复计算,其综合性能低于基于 delta 表的合并算法;

(6) 根据算法性能测试,我们设置了 3 个性能阈值:ζ表示假设更新复杂度,$\lambda=|cuboid|/|\boldsymbol{B}|$,$\mu$表示因 what-if 更新所影响到的 delta 记录与基表记录的比值,即$\mu=|\boldsymbol{D}|/|\boldsymbol{B}|$.当采用查询重写算法时,查询性能与假设更新的类型、假设更新数据子空间在整个数据空间中的分布以及假设更新所影响的记录数量相关.不同的算法性能受一个或多个阈值的影响,在不同的数据环境下会产生不同的性能指标,在系统设计时可以根据系统参数与阈值之间的关系来确定算法的选择策略.阈值的确定受数据库 SQL 引擎对不同类型查询处理性能的影响,需要根据具体的数据库系统进行测试并设置.

6 What-If analysis 查询处理策略

通过实验,我们全面对比并分析了不同的 what-if analysis 查询处理算法的性能并分析了算法性能的影响因素,我们以算法的时间复杂度作为代价分析的指标,具体分析不同算法的执行代价,并以此为基础给出 what-if analysis 的代价评估模型,从而为系统查询算法选择策略提供依据.

6.1 delta表合并算法代价模型

假设分析查询执行代价包括假设数据视图合并的代价和 *group-by* 操作的代价,具体分析如下:

$$cost(fullJoinBDM)=cost(\boldsymbol{B}\text{ full join }\boldsymbol{D})+cost(generate\ view)+cost(group\text{-}by),$$

$$cost(leftJoinBDM)=cost(\boldsymbol{B}\text{ left join }\sigma_{FLAG='U'\text{ or }FLAG='D'}(\boldsymbol{D}))+cost(generate\ view)+cost(UNION\ \sigma_{FLAG='I'}(\boldsymbol{D}))+cost(group\text{-}by),$$

$$cost(deltaMap)=cost(scan\ deltaMap)+cost(generate\ view)+cost(group\text{-}by),$$

$$cost(pre\text{-}mergeBDM)=cost(\boldsymbol{B}\ UNION\ \boldsymbol{D})+cost(group\text{-}by).$$

在实际的查询执行过程中,SQL 引擎会将部分查询处理阶段转换为流水处理过程,因此,总的查询执行时间小于各执行阶段执行时间的总和.

6.2 查询重写算法代价模型

在查询重写算法中,假设分析查询执行代价包括基于查询重写模式的假设数据视图合并的代价和 *group-by* 操作的代价,具体分析如下:

$$cost(re\text{-}writing\ query)=cost(what\text{-}if\ update)+cost(what\text{-}if\ delete)+cost(what\text{-}if\ insert)+cost(UNIONs)+cost(group\text{-}by).$$

6.3 Δcuboid合并算法代价模型

Δcuboid 合并算法的执行过程分为两个阶段:(1) 产生查询对应的 cuboid 或Δcuboid;(2) 执行 *group-by* 操作,具体分析如下:

$$cost(\Delta cuboid)=cost(generate\ cuboid)+cost\ (generate\ delta\ set)+cost(generate\ \Delta cuboid)+cost(\Delta cuboid\ merge)+cost(group\text{-}by).$$

© 中国科学院软件研究所 http://www.jos.org.cn

6.4 基于算法时间复杂度的代价评估模型

磁盘数据库的代价评估模型是以磁盘块访问的数量作为代价评估指标,在内存数据库中,查询处理的代价包括 CPU 有效计算时间、各级缓存产生的延迟时间以及控制指令和资源缺失所导致的延迟时间.目前,L2 cache 在数据访问时的命中率一般在 80%以上,通常采用 LRU(least recently unused)替换算法,但其存储容量相对较小,高端的 CPU 大约在十几 MB 左右.在当前的技术水平下,还不能把数据库中成熟的缓冲区管理算法扩展到 cache 上进行数据访问调度管理.因此,我们主要以 CPU 的有效计算时间作为算法代价评估的基础,在具体分析中采用时间复杂度模型来计算算法执行的时间代价.

1) 查询处理子操作的时间复杂度

首先分析前面所描述的各类算法中不同的处理子操作的时间复杂度.我们规定$|\boldsymbol{B}|=n$,$|\boldsymbol{D}|=m$,索引查询的时间复杂度为 $O(\log_2 n)$,排序操作的时间复杂度为 $O(n\log_2 n)$,*group-by* 操作的时间复杂度为 $O(n\log_2 n+n)$,即:

$$cost(\boldsymbol{B}\ full\ join\ \boldsymbol{D})\approx cost(\boldsymbol{B}\ \text{left join}\ \boldsymbol{D})\approx cost(\boldsymbol{B}\ \text{join}\ \boldsymbol{D})=O(|\boldsymbol{B}|\times|\boldsymbol{D}|)=O(m\times\log_2 n),$$

$$cost(generate\ view)=O(|view|)=O(n),$$

$$cost(group\text{-}by)=O(n\log_2 n+n),$$

$$cost(\boldsymbol{B}\ UNION\ \boldsymbol{D})=O(n+m),$$

$cost(\sigma_A(\boldsymbol{B}))=O(r\times n)$,$r$ 为选择操作的选择率.

2) what-if anaysis 查询执行代价

如表 1 所示,我们通过算法的时间复杂度来衡量不同算法的实现代价.

Table 1 Cost of what-if analysis algorithms

表 1 what-if analysis 算法的执行代价

Algorithms	Cost of execution
fullJoinBDM	$O(m\times\log_2 n+(n+30\%m)\times(2+\log_2(n+30\%m)))$
leftJoinBDM	$O(2n+70\%m+(n+30\%m)\times(2+\log_2(n+30\%m)))$
deltaMap	$O((n+30\%m)\times(2+\log_2(n+30\%m)))$
pre-mergeBDM	$O(n+m+(n+30\%m)\times(1+\log_2(n+30\%m)))$
re-writing query	$O(n(1.3+0.4\alpha+0.3\beta+0.3\gamma)+(n+30\%m)\times(1+\log_2(n+30\%m)))$ Note: α, β, γ are weighted values for hypothetical *update*, *delete* and *insert* operations
Δcuboid (delta table) with materialized view	$O(m\times(1+\log_2 m)+\lambda n\times\log_2(\lambda\omega n))$ Note: $\lambda=\|cuboid\|/\|\boldsymbol{B}\|$, $\omega=\|\Delta cuboid\|/\|cuboid\|$,
Δcuboid (delta table) without materialized view	$O(n\times(1+\log_2 n)+m\times(1+\log_2 m)+\lambda n\times\log_2(\lambda\omega n))$
Δcuboid (re-writing query) with materialized view	$O((0.4\alpha+0.3\beta+0.3\gamma)m+m\times(1+\log_2 m)+\lambda n\times\log_2(\lambda\omega n))$
Δcuboid (re-writing query) without materialized view	$O(n\times(1+\log_2 n)+(0.4\alpha+0.3\beta+0.3\gamma)m+m\times(1+\log_2 m)+\lambda n\times\log_2(\lambda\omega n))$

3) 基于 what-if analysis 代价评估模型的查询选择策略

what-if analysis 具有复杂的应用背景,不同的实现算法也有不同的应用场景和数据场景,在设计原型系统时,需要根据应用环境的需求开发出不同的算法实现模块,通过查询代价评估与查询选择策略来实现具体的查询处理.

查询选择策略受以下几个因素的影响:

- 支持度:不同的 what-if analysis 查询实现算法具有各自的应用范围,对聚集函数的支持程度各不相同;
- 系统约束:不同的实现算法有不同的时间与空间开销,当系统对时间或空间开销具有倾向性时,需要优先选择满足系统约束条件和需求的算法;
- 性能需求:在算法可用性和用户需求的可满足性相同的情况下,根据代价评估模型优先选择性能较好的查询实现算法.

在进行系统的算法选择时,综合性能指标为 0 表示该算法不能在当前的应用场景中使用.在进行算法选择评估时,首先根据算法对应用需求的支持度和系统约束条件排除不能使用的算法,然后再根据系统开销策略和视图支持机制选择出候选算法集合,然后根据应用场景中的数据分布状况、假设更新的数据集特征、查询结果集的数据特征等影响因素对算法进行全面的代价评估,最后选择适合于应用需求的最佳算法.各项的权值为该

© 中国科学院软件研究所 http://www.jos.org.cn

算法的执行时间代价指标,可以根据查询的数据特征和查询算法实时计算出来,也可以根据查询执行过程的统计信息以经验值代替.在示例中,我们在算法复杂度分析的基础上采用经验值来表示各算法在不同数据场景下的性能指标值,也可以将经验值作为系数,根据算法的执行代价精确地计算出各算法的性能指标,将加权的算法性能指标作为依据作进一步的计算和评估.

如表 2 所示,其中,3,2,1,0 表示不同的支持度,值越大,支持度越高,0 表示不支持;ζ表示假设更新复杂度;λ表示 cuboid 中记录数量与基表记录数量的比例,记作λ=|*cuboid*|/|*B*|;μ表示 delta 表中记录数量与基本记录数量的比例,记作μ=|*D*|/|*B*|.当按照应用的需求和数据场景参数进行算法选择时,将对应的各行指标值相乘,求出各算法的综合指标值,然后选择具有最佳的指标算法或在 TOP *N* 个候选算法中再通过人工决策来选择最优的执行算法.例如,查询处理的数据场景为:计算复杂度高、空间开销小、查询结果为低基数的查询结果集.计算方法为:将表中第 7 行、第 8 行、第 10 行对应的数值相乘,结果为{3 3 6 6 3 9 18 8 6}.其中,综合指标最高的是基于 delta 表的、无物化视图支持的Δcuboid 合并算法.

Table 2 Selection policy of what-if analysis algorithms

表 2 what-if analysis 算法的选择策略

			fullJoinBDM	leftJoinBDM	deltaMap	pre-mergeBDM	re-writing query	Δcuboid (deltaT) with MV	Δcuboid (deltaT) no MV	Δcuboid (re-wQ) with MV	Δcuboid (re-wQ) no MV
Support	Multi-Version multi-user		3	3	3	3	1	2	2	1	1
	SUM, COUNT, AVG		3	3	3	3	3	3	3	3	3
	Other aggregate functions		3	3	3	0	3	0	0	0	0
System constrains	Materialized views	Yes	—	—	—	—	—	3	—	3	—
		No	1	1	1	1	1	—	2	—	2
	Space cost	Large	3	3	3	3	3	3	3	3	3
		Small	1	1	2	2	3	1	2	2	3
Performance	ζ	High	3	3	3	3	1	3	3	2	1
		Low	3	3	3	3	3	3	3	3	2
	λ	Large	2	2	2	2	2	2	1	2	1
		Small	1	1	1	1	1	3	3	2	2
	μ	Large	1	1	2	2	3	2	1	2	1
		Small	2	2	3	3	3	3	2	2	1

7 结束语

本文系统地讨论并扩展了 what-if analysis 的相关算法,详细分析了各种算法的执行过程和执行效率,设计了基于 SQL 平台的统一的性能测试实验.通过实验获得了各种算法在相同性能度量指标下的性能对比数据,从而可以量化地分析不同算法在应用中的性能特性.提出了基于内存数据库 what-if OLAP 的性能评估模型,并根据性能评估模型设计了查询选择策略,提高了 what-if OLAP 对应用环境的适应性.

本文研究的重点是各种算法的性能分析与对比,对多版本模式下的各种算法的性能特征未作深入的分析和研究.在多用户、多版本的应用场景中,what-if analysis 是一个多用户协作、多阶段操作和数据共享的过程,假设更新往往带有很多的继承性,独立的假设更新与级联的假设更新在不同的实现算法中有不同的性能特征,为系统算法的代价评估和选择增加了新的影响因素.我们下一步的工作是基于多用户、多版本模式下的 what-if analysis 查询处理研究.

References:

[1] Stonebraker M, Keller K. Embedding expert knowledge and hypothetical data bases into a data base system. In: Chen PP, Sprowls RC, eds. Proc. of the ACM-SIGMOD'80 Int'l Conf. on Management of Data. New York: ACM Press, 1980. 58−66.

© 中国科学院软件研究所 http://www.jos.org.cn

[2] Stonebraker M. Hypothetical data bases as views. In: Lien YE, ed. Proc. of the ACM-SIGMOD'81 Int'l Conf. on Management of Data. New York: ACM Press, 1981. 224−229.

[3] Woodfill J, Stonebraker M. An implementation of hypothetical relations. In: Schkolnick M, Thanos C, eds. Proc. of the 9th Int'l Conf. on Very Large Data Base. San Francisco: Morgan Kaufmann Publishers, 1983. 157−166.

[4] Wang S, Xiao YQ, Zhang YS, Chen H. Research on OLAP system supporting what-if analysis. Chinese Journal of Computers, 2008,31(9):1573−1587 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=2685 [doi: 10.3724/SP.J.1016.2008.01573]

[5] Zhang YS, Xiao YQ, Xu F, Zhou GL, Wang S, Chen H. Research on storage policy in main memory database based on what-if analysis. Journal of Computer Research and Development, 2008,45(10):136−141 (in Chinese with English abstract).

[6] Balmin A, Papadimitriou T, Papakonstantinou Y. Hypothetical queries in an OLAP environment. In: Abbadi AE, Brodie ML, eds. Proc. of the 26th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2000. 220−231.

[7] Mumick IS, Quass D, Mumick BS. Maintenance of data cubes and summary tables in a warehouse. In: Peckham J, ed. Proc. of the ACM-SIGMOD Conf. on Management of Data. New York: ACM Press, 1997. 100−111.

[8] Lee KY, Kim M. Efficient incremental maintenance of data cubes. In: Dayal U, Whang KY, eds. Proc. of the 32nd Int'l Conf. on Very Large Data Bases. New York: ACM Press, 2006. 823−833.

[9] Witkowski A, Bellamkonda S, Bozkayz T, Naimat A, Sheng L, Subramanian S, Waingold A. Query by Excel. In: Böhm K, Jensen CS, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. New York: ACM Press, 2005. 1204−1215.

[10] Witkowski A, Bellamkonda S, Bozkayz T, Folkert N, Gukpta A, Sheng L, Subramanian S. Business modeling using SQL spreadsheets. In: Freytag JC, Lockemann PC, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases. New York: ACM Press, 2003. 1117−1120.

[11] Timothy G, Richard H. A framework for implementing hypothetical queries. In: Peckha J, ed. Proc. of the ACM-SIGMOD'97 Int'l Conf. on Management of Data. New York: ACM Press, 1997. 231−242.

[12] Ramirez RG, Kulkarni UR, Moser KA. The cost of retrievals in what-if databases. In: Proc. of the 24th Annual Hawaii Int'l Conf. on System Sciences. IEEE Press, 1991. 136−145. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=183973

[13] Valduriez P. Join indices. ACM Trans. on Database Systems , 1987,12(2):218−246. [doi: 10.1145/22952.22955]

附中文参考文献:

[4] 王珊,肖艳芹,张延松,陈红.支持 What-if 分析的 OLAP 系统研究.计算机学报,2008,31(9):1573−1587. http://cjc.ict.ac.cn/qwjs/view.asp?id=2685 [doi: 10.3724/SP.J.1016.2008.01573]

[5] 张延松,肖艳芹,徐凡,周国亮,王珊,陈红.基于 what-if 分析的内存数据库存储策略研究.计算机研究与发展,2008,45(10):136−141.

张延松(1973－),男,山东泰安人,博士生,副教授,主要研究领域为高性能数据库,内存数据库,OLAP 应用.

王珊(1944－),女,教授,博士生导师,CCF 高级会员,主要研究领域为高性能数据库,数据库与信息检索,内存数据库,视频数据库.

肖艳芹(1974－),女,博士,讲师,主要研究领域为高性能数据库,内存数据库,OLAP 应用.

陈红(1965－),女,教授,博士生导师,CCF 高级会员,主要研究领域为数据仓库与数据挖掘,传感器数据管理.

© 中国科学院软件研究所 http://www.jos.org.cn