软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(4):753−767 [doi: 10.13328/j.cnki.jos.004568]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一个基于三元组存储的列式 OLAP 查询执行引擎*

朱阅岸[1,2], 张延松[1,2,3], 周 烜[1,2], 王 珊[1,2]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),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)

通讯作者: 朱阅岸, E-mail: iwillgoon@ruc.edu.cn

摘 要: 大数据与传统的数据仓库技术相结合产生了大数据实时分析处理需要(volume+velocity),它要求大数据背景下的数据仓库不能过多地依赖物化、索引等高存储代价的优化技术,而要提高实时处理能力来应对大数据分析中数据量大、查询分析复杂等特点.这些查询分析操作一般表现为在事实表和维表之间连接操作的基础上对结果集上进行分组聚集等操作.因此,表连接和分组聚集操作是 ROLAP(relational OLAP)性能的两个重要决定因素.研究了新硬件平台下针对大规模数据的 OLAP 查询的性能,设计新的列存储 OLAP 查询执行引擎 CDDTA-MMDB (columnar direct dimensional tuple access-main memory databasequeryexecutionengine,直接维表元组访问的内存数据库查询执行引擎).基于三元组的物化策略,使得 CDDTA-MMDB 能够减少内存列存储模型上表连接操作访问基表和中间数据结构的次数.首先,CDDTA-MMDB 将查询分解为作用在维表和事实表上的子查询,如果只涉及过滤操作,子查询将生成〈代理键,布尔值〉二元组;否则,子查询生成〈代理键,关键字,值〉三元组.然后,只需一趟扫描事实表,利用事实表的外键映射函数直接定位相应三元组或者二元组,完成相应的过滤、连接或聚集操作.CDDTA-MMDB 充分考虑了内存列存储数据库的设计原则,尽量减少随机内存访问.实验结果表明:CDDTA-MMDB 是高效的,与具代表性的列存储数据库相比,比 MonetDB 5.5 快 2.5 倍,比 C-store 的 invisible join 快 5 倍;并且,CDDTA-MMDB 在多核处理器上具有线性加速比.

关键词: 大数据分析;联机分析处理;内存列存储数据库;表连接算法;物化策略

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 朱阅岸,张延松,周烜,王珊.一个基于三元组存储的列式 OLAP 查询执行引擎.软件学报,2014,25(4):753−767. http://www.jos.org.cn/1000-9825/4568.htm

英文引用格式: Zhu YA, Zhang YS, Zhou X, Wang S. Column-Oriented query execution engine for OLAP based on triplet. Ruan Jian Xue Bao/Journal of Software, 2014,25(4):753−767 (in Chinese). http://www.jos.org.cn/1000-9825/4568.htm

Column-Oriented Query Execution Engine for OLAP Based on Triplet

ZHU Yue-An[1,2], ZHANG Yan-Song[1,2,3], ZHOU Xuan[1,2], WANG Shan[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)

Corresponding author: ZHU Yue-An, E-mail: iwillgoon@ruc.edu.cn

* 基金项目: 国家科技重大专项(核高基)(2010ZX01042-001-002); 国家自然科学基金(61272138, 61232007); 中国人民大学研究生科学研究基金(13XNH216)

收稿时间: 2013-10-13; 修改时间: 2013-12-18; 定稿时间: 2014-01-27

Abstract: Integrating big data and traditional data warehouse (DW) techniques bring demand for real-time big data analysis. The new demand means DW can not depend too much on the optimization such as materialization and indexing which consume large space, but instead needs to enhance ability of real-time analysis to handle big data analysis which usually issues complex queries on huge data volumes. Those queries usually consist in applying group or aggregation operator on the join result between fact table and dimension table(s). The join and group operation often are the bottle-necks for performance improvement. This paper studies the OLAP performance under the new hardware platform and big data environment, and develops a new OLAP query execution engine in columnar storage, called CDDTA-MMDB (columnar direct dimensional tuple access for main memory database query execution engine). The optimized materialization makes CDDTA-MMDB reduce access to base table and intermediate data structure during join procedure. CDDTA-MMDB decomposes the query into sub-queries on the fact table and dimension table respectively. If the sub-query on dimension table only serves as filter, it will produce the binary tuple ⟨*surrogate*,*Boolean_value*⟩; otherwise, it will produce the triplet in the form of ⟨*surrogate*,*key*,*value*⟩. Thus, by just scanning the fact table one-pass and utilizing the mapping function of foreign keys in fact table to directly access the binary tuples or triplets, the executor can accomplish the join, filter and group operations. Consideration is fully placed on the design principle for the main-memory columnar database. Experimental results show that the system is efficient and can be 2.5 times faster than MonetDB 5.5 and 5 times faster than invisible join used by C-store. Moreover, it scales linearly on multi-core processors.
Key words: big data analysis; OLAP; main-memory columnar database; join algorithm; materialization

大数据在2001年最早提出时有3个维度:volume(数据量)、velocity(数据产生和输出的速度)以及variety(不同数据类型和来源),2012年,Gartner更新了大数据的概念,其中一个重要的变化是high volume,high velocity, and/or high variety代替了原来的3个固有维度[1],variety成为可选维度.这种变化表明:大数据已逐渐与各个领域相结合,而volume与velocity的结合则产生了大数据仓库需求,大数据上的OLAP正日益成为大数据分析的一个重要应用领域.与事务型数据库的查询相比,DW(data warehouse)的查询(OLAP查询)通常很复杂、涉及的元组属性较少,且主要是以读为主.通常,OLAP查询所需处理时间较长,而在大规模数据上的OLAP查询让这一矛盾显得更加突出.由于列存储数据库只访问与查询相关的数据,因此它能高效地利用内存带宽和高速缓存、减少磁盘I/O次数.这一特性使得列存储系统在应对海量数据上的OLAP查询带来的挑战时具有很大优势,C-store,SyBase IQ和MonetDB[2,3]等列存储系统都验证了列存储在应对读密集型应用的优越性能.为了改进OLAP查询的性能,微软也已将列存储加入到SQL Server 2012[4]之中.随着内存容量的增大以及其价格的下降,涌现出一大批性能卓越的内存列存储数据库,MonetDB[2,5]就是其中的典型.SAP HANA数据库[6,7]是高性能的混合存储内存数据库.当处理分析型的工作负载时,它采用列存储的数据管理策略.这些系统的性能在特定工作负载下,尤其是在像DW中遇到的读密集型的工作负载情况下,比传统的行存储数据库要高出一个数量级以上.

为了克服OLAP查询中的多表连接和分组聚集的性能瓶颈,以应对大规模数据分析带来的挑战,本文介绍我们开发的一个内存列存储查询执行引擎:CDDTA-MMDB.我们的目标是进行实时的大规模数据分析,它能够以平均1.5s左右的时间完成对100GB的SSB(star schema benchmark)测试数据集的查询.此外,CDDTA-MMDB可以部署在分布式环境下,利用反转星型的处理模式[8]轻松应对TB甚至是PB级的海量数据分析.这个系统是我们实验室在研究大数据背景下一系列研究成果[8−10]之一,主要贡献体现在以下几个方面:

(1) 存储模型:CDDTA-MMDB采用分解存储模型(decomposed storage model,简称DSM),垂直划分关系表.我们扩展MonetDB的二元关联表(binary association table,简称BAT),维护一个三元组⟨代理键,关键字,值⟩或二元组集合的表.这种扩展的数据类型除了能更好地利用高速缓存块以外,也是轻量级物化的关键数据结构;

(2) 查询处理:CDDTA-MMDB运用分治策略处理OLAP查询.首先,将查询分解为作用在维表和事实表上的子查询;然后,根据子查询的特征改写子查询,以生成三元组或者二元组;最后,通过一趟扫描事实表,利用事实表的外键映射函数定位相应的三元组或者二元组,完成过滤、连接或者分组聚集操作;

(3) 列存储数据库的物化策略:在系统里提出一种新的物化策略,称为轻量级物化.轻量级物化与延迟物化都是将物化操作在查询计划里尽量推迟,但是轻量级物化能够减少内存的访问次数,加速OLAP查询中常见的分组聚集等操作;

(4) 实验:我们在 SSB 上使用标准测试数据和标准测试查询进行了全面的实验,测试系统在多核处理器环境下的扩展性;测试轻量级物化的性能;测试系统的总体性能.实验结果表明:CDDTA-MMDB 基本上达到了线性加速比;在星型模型数据库仓库的标准测试数据集下,比 MonetDB 5.5 快 2.5 倍,比 C-store 的针对星型模型数据仓库的查询执行引擎(invisible join)快 5 倍.

本文第 1 节介绍相关工作.第 2 节介绍系统总体架构,包括存储管理、查询解析、查询处理以及优化.第 3 节是实验部分.最后一节是总结及未来的工作.

1 相关工作

尽管将数据库关系表垂直划分以改进性能的概念很早就已经提出[11],MonetDB 和 MonetDB/X100[2,3,5,12]是设计列存储系统和向量执行引擎的先驱.MonetDB 和 MonetDB/X100 展示出:列存储由于 CPU 和高速缓存的有效利用能够在以读为主的应用场景中远优于传统的行数据库.MonetDB 的设计、体系结构和实现都重新审视了传统数据库的各个部件,以充分挖掘现代计算机的潜能.它不仅采用列存储逻辑组织数据,而且提供全新的针对列存储的执行引擎,开发高速缓存敏感的数据结构和优化的算法,以最优地利用分级存储体系结构.MonetDB 前端负责将用户的数据模型转换成二元关联表(binary association tables,简称 BATs)和将用户输入的查询编译成 MonetDB 汇编语言(MonetDB assembly language,简称 MAL)[5].MonetDB 将关系表垂直划分,每个 BAT 由形如⟨*surrogate*,*value*⟩的二元组构成,二元组的 *value* 为定长数据值或者为变长数据的偏移量.一个具有 k 个属性的 MonetDB 关系表会被映射成 k 个 BAT 表.为了方便元组重构,元组 t 的所有属性都在 BAT 的同一位置.MAL 的核心由作用在 BAT 上的底层关系代数构成.行关系代数查询执行计划被转换成 BAT 代数,并且被编译成 MAL 程序.这些 MAL 程序会被以“operator-at-a-time”的方式评估,也就是在激发后续的数据依赖的操作之前在整个输入数据集上评估这些操作.每个 BAT 算数操作符都会被映射成简单的 MAL 指令.每一个复杂的操作都被分解成一系列 BAT 代数操作符,每一个代数操作符在整列数据上执行简单的操作(“块处理”),减少了函数调用开销.

C-store[13]是近些年来的另外一个列存储系统,它拥有许多与 MonetDB/X100 一样的特性,包括针对直接作用在压缩数据上的操作的优化.C-store 支持标准的关系模型,其查询语言为 SQL.在物理存储层,C-store 没有采取传统行存储,也不像 MonetDB 那样对每个属性单独维护一个 BAT,而是将一个逻辑表在一个或者多个属性上垂直投影(project).系统维护这些投影,以应对用户查询.利用连接索引,C-store 可以将这些投影恢复成原来的表.例如,为了重构表 T,只需从表的一个覆盖集 $T_1,T_2,...,T_k$ 中找到一个连接路径即可.在查询优化方面,C-store 采用基于代价的 Selinger 风格[14]的优化,利用两阶段优化来限制查询计划搜索空间[15].为了应对在海量数据上的 OLAP 查询,特别是针对星型模型的数据仓库的查询,C-store 引进 invisible join[16]算法.Invisible join 能够改进在列存储数据库上星型模型的表之间主键/外键模式的表连接性能.

MonetDB 与 C-Store 都是列存储数据库,因此,它们不可避免地需要在查询计划的某个点将逻辑上属于同一元组的列值合并起来,这个过程就是物化.按照物化时机来分类,可以将物化策略分为早物化与延迟物化[17],MonetDB/X100 使用延迟物化策略[2].然而,MonetDB 中的位置描述符(也叫作选择向量)与列值分开存放,并且位于高速缓存的数据是以非压缩的形式存在的,这会损失直接在压缩数据上进行操作所带来的性能优化.C-store 采用的是并行延迟物化策略,也就是并行物化前的操作,例如生成位置描述符.后来,C-store 改进了物化策略,采用启发式算法来决定是采用早物化或者是延迟物化[17].文献[18,19]深入对比了行存储与基于早物化的列存储.

受益于内存容量的快速增长与价格的进一步下降,不仅事务型数据库可以将大部分工作集加载到内存,而且分析型数据库也朝这个方向发展,产生了实时分析型数据仓库[2,6,7,13].内存数据库消除了昂贵的磁盘 I/O、数据页格式在内存与磁盘之间的转换操作以及复杂的缓冲区管理[20,21].相比磁盘数据库,内存数据库在性能上有很大的提升.

CDDTA-MMDB 是一个内存列存储数据库查询执行引擎,是针对星型模型数据仓库上的 OLAP 而提出来的解决方案.它基于三元组改进了列存储中的物化方式,避免了早物化与延迟物化的缺点(早物化会构建无用元

组而延迟物化需要重复读取数据块);并且,CDDTA-MMDB 改进了 invisible join 算法,能够一趟扫描事实表完成表连接.其查询解析算法是基于分解的思想.

2 CDDTA-MMDB 总体架构

CDDTA-MMDB 是一个内存列存储数据库执行引擎,其主要模块如图 1 所示.应用程序和用户发送查询请求到系统;查询解析模块将查询分解成作用在维表与事实表上的一系列子查询,这些子查询所涉及的表互不相交.查询解析算法采用分治策略的主要原因是,维表上的子查询与事实表上的子查询需要以不同的方式处理:维表上的操作主要是提供过滤器和分组值,作用在维表上改写后的子查询由系统动态地生成三元组或者是二元组,如图 1 中虚线矩形所示;而事实表上的查询信息会被系统解析、存储,利用这些信息,查询执行器知道需要访问事实表上哪些列.目前,查询解析模块的语义检查只是简单地查找系统目录,确定查询中出现的每个属性是否合法.查询优化器生成查询执行计划,这个查询执行计划的生成,依赖于维表与事实表上的过滤条件.查询执行器的主要操作是表扫描.在星型模型数据库仓库上,事实表与维表存在主外键关系,且表连接操作主要是在它们之间进行.根据这个特点,CDDTA-MMDB 将表连接操作转换成事实表上外键的内存定位操作.扫描一趟事实表,即可完成表连接、过滤,聚集操作.数据文件在系统启动的时候被加载到内存.下面详细地介绍系统的各个模块.

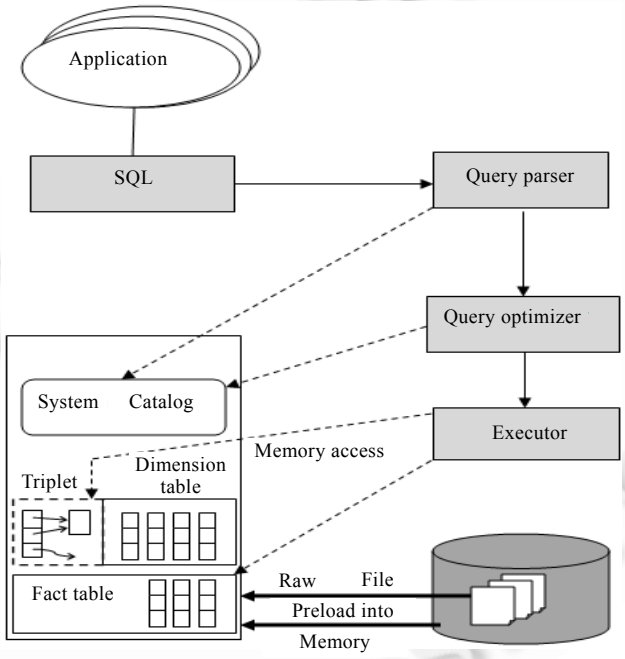


Fig.1 CDDTA-MMDB architecture overview

图 1 CDDTA-MMDB 总体架构

2.1 维表/事实表的管理

如上所述,系统将查询分解为作用在事实表和维表上的子查询.作用在维表上的子查询会被重写,以产生三元组⟨*surrogate*,*key*,*value*⟩或者是二元组⟨*surrogate*,*Boolean_value*⟩(当子查询只有过滤作用的时候,系统将为整个子查询产生二元组).在三元组中,一个 *key* 会对应多个 *value*,亦即 *key* 与 *value* 是一对多的关系.MonetDB 的二元关联表中,*value* 要么为定长数据,要么为变长数据的偏移量.本文扩展了 MonetDB 的二元组中 *value* 的含义,使它包含过滤信息(我们的系统中,三元组的 *surrogate* 与 MonetDB 二元组中的 *surrogate* 都不实际存储).在 CDDTA-MMDB 中,*key* 的作用类似于 MonetDB 中二元组的 *value*,它具有两层含义:

① 系统通过它可以知道在该位置上的元组是否满足过滤条件;

② 它是该属性的某个数据的 ID,系统可以在后续的查询计划,例如聚集、排序操作中直接利用这个 ID.

这个设计是实现系统轻量级物化的关键.系统将三元组的 *key* 存放于地址连续的内存空间,称为 *key vector*.*key* 可以是其对应 *value* 的哈希值,也可以是 *value* 的内存地址,因此,三元组的 *key* 可以看成是其对应维表列值的一种轻量级数据压缩形式.系统在查询执行中直接对以整型存在的 *key* 进行操作,而不需要处理变长数据与复杂数据类型,从而加速查询执行操作,提高系统响应时间.同时,我们在 *key* 值上加入过滤信息,使得满足过滤条件的 *key* 值不为 0.三元组的设计可以使得 CDDTA-MMDB 的物化策略避免延迟物化需要重复读取数据块的缺点与维护中间数据结构的代价.三元组的 *value* 属性只在输出结果时才会被使用.此外,可以通过事实表外键的映射函数直接定位维表相应元组,如同定位数组元素的操作.三元组或者是二元组与其对应的维表元组一一对应,因此可以通过一趟扫描事实表完成过滤、连接和分组聚集操作.图 2 给出了三元组的一个示例:假如某个维表的一个非主属性列包含 4 个属性值{Sam,Bob,Bob,John},满足某个查询条件的属性值只有{Sam,Bob},则满足过滤条件的属性值相应的 *key* 值不为 0.注意,*key vector* 中元素的位置与维表中元组的位置一一对应.系统可以在查询计划的各个阶段直接利用该 *key* 值,完成过滤、聚集等操作.

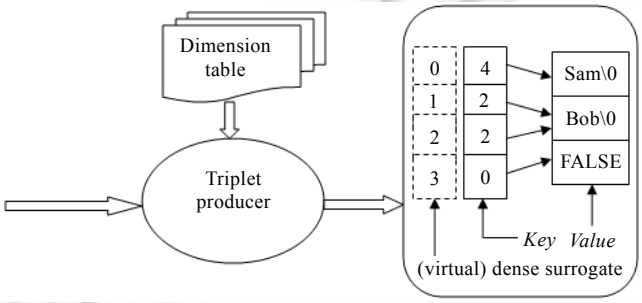


Fig.2 An example of triplets

图 2 三元组示例

2.2 查询分解与三元组的生成

在 CDDTA-MMDB 中,采用分治策略处理用户的查询.在系统中,查询被划分成独立的子查询,这些子查询所涉及的表互不相交.根据用户查询的目标属性和过滤条件,这些子查询会被改写以产生三元组.下面详细介绍查询分解和查询改写算法.

查询分解算法首先由文献[22,23]提出,它是为数据库系统 INGRES 的查询语言 QUEL 而开发的,给出 INGRES 中处理多参数查询的策略.INGRES 将查询分解成一系列单参数的做法是:

a) 归约:将查询的各个部分分解成包含单个参数的子查询;

b) 元组替换:每次用元组替换多参数查询中的某个参数.

算法的不足是,当表之间出现“链接”信息时,分解的原子语句会包含两个变量.在我们的应用场景中,表之间的“链接”信息可以通过主键/外键的对应关系消解.因此,我们的分解算法不会出现包含两个变量的子查询的情况.我们设计的查询分解算法流程如下:首先扫描 *select* 句子的目标属性列表;同时,通过查找系统目录确定属性列表的各个变量所属的表;下一步,抽取在各个表上的过滤条件,并记录“链接”信息.“链接”信息在扫描事实表时确定哪些维表需要访问.图 3 给出了 SSB 中 Q4.1(查询 4.1)的 SQL 部件,它可以划分为 *select* 句子目标属性列表(target list)、链接信息(link information)、过滤条件(qualification)和分组器(grouper)等.以 Q4.1 为例,扫描目标属性列表之后通过查找系统目录,可以获得以下信息:

$$d\_year \in dwdate, c\_nation \in customer, (lo\_revenue, lo\_suppcost) \in lineorder.$$

我们利用有限状态自动机(deterministic finite automaton,简称 DFA)分解过滤条件.如图 3 所示,在星型模型

中,事实表和维表通过主键/外键关联起来.因此,事实表中的外键可以作为连接索引直接定位维表的元组.在这种情况下,维表与事实表的“链接”可以消解.取而代之的是,只需让系统记录扫描事实表时需要访问哪些维表.因此,本文的查询分解算法能够保证在这个应用场景下只有单变量的子查询.在获取目标属性列表和分解作用在各个表上的过滤条件之后,就可以改写子查询了.改写子查询需要分 3 种情况分别加以考虑:

1) 存在作用在表上的过滤条件,但是该表的属性没有出现在 *select* 目标属性列表(target list)中.

这种情况下,系统只需为该改写后的 SQL 语句产生布尔值.因此,改写后的查询语句与下面类似:

*RQ*1:SELECT CASE WHEN (*p_mfgr*=‘MFGR#1’ or *p_mfgr*=‘MFGR#2’) THEN 1 ELSE 0 END FROM *part*

*RQ*2:SELECT CASE WHEN *s_nation*=‘AMERICA’ THEN 1 ELSE 0END FROM *supplier*

2) 存在作用在表上的过滤条件,同时,该表的属性也出现在目标属性列表中.

这时候,需要返回满足过滤条件的元组.对于不符合条件的元组,返回 0,同时标记该位置上的元组不符合过滤条件.改写后的 SQL 语句类似以下的结构:

*RQ*3:SELECT CASE WHEN *c_region*=‘AMERICA’ THEN *c_nation* ELSE 0 END FROM *customer*

3) 没有作用在该表上的 SQL 上限值条件,但是该表的某些属性出现在目标属性列表中.

如果出现了这种情况,则需要返回表上该属性的所有数据.改写后的 SQL 语句类似以下结构:

*RQ*4: SELECT CASE WHEN *TRUE* THEN *d_year* ELSE 0 END FROM dwdate

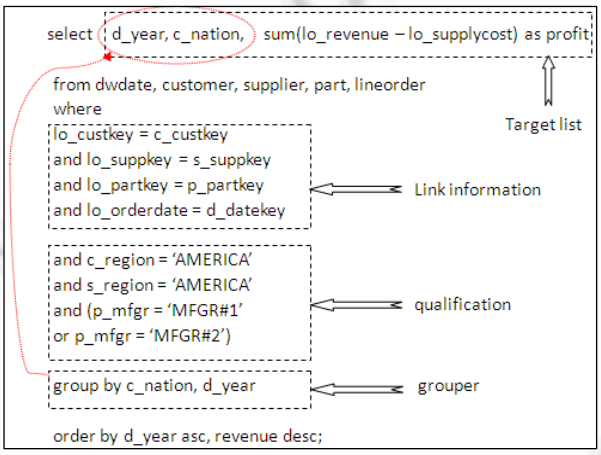


Fig.3 SQL query components in Q4.1

图 3 Q4.1 的 SQL 查询部件

例程 *tripletProducer* 给出了产生三元组的算法:

- 首先把游标定位在将要处理的维表的第 1 条记录,如果记录不符合过滤条件,则直接将 *key* 值赋为 0,并把 *key* 加入 *key vector*;
- 否则,判断与该条记录相同的值是否存在:如果存在,则首先获取该属性的 *key* 值,并把 *key* 加入 *key vector*;如果不存在,则将该记录放入哈希表,增加 *key* 值并把它放入 *key vector*.

算法 1. *tripletProducer*.

输入:查询涉及的维表、改写后的 SQL;

输出:三元组或者二元组.

```
begin
  for each dimension table involved in query and corresponding rewritten SQL on it do {
    locate the cursor on the first record according to rewritten SQL;
    while cursor is not to end do {
      get attribute from result set;
```

```
    If attribute is FALSE according to rewritten SQL {     //维表上该位置的属性值不满足过滤条件,
      assign 0 to key;                                     //该属性对应的 key 值为 0
      push back key into the key vecotor;
      cursor moves to next record; continue;
    }//end if
    use the attribute to probe the hash table;
    if attribute not in the hash table {                   //维表上该属性满足过滤条件
      increment key;                                       //且目标属性尚且不在哈希表中,生成不同的 key 值
      insert the attribute into hash table;
    }else {          //维表上该属性满足过滤条件,且目标属性已在哈希表中,返回已经存在的key值即可
      get the key of the attribute;
    }//end if
    push back key into the key vector;                     //将对应目标属性 key 值存放于 key 值向量中
    cursor moves to next record;
  }//end while
 } //end for
end tripletProducer
```

2.3 基于三元组的轻量级物化

以下符号按照文献[24]来定义:

- σ:选择操作,对应 SQL 语句 WHERE 条件;
- π:投影操作,对应 SQL 语句 SELECT 语句;
- γ:分组聚集操作,对应 SQL 语句 GROUP BY.

另外,增加如下操作:

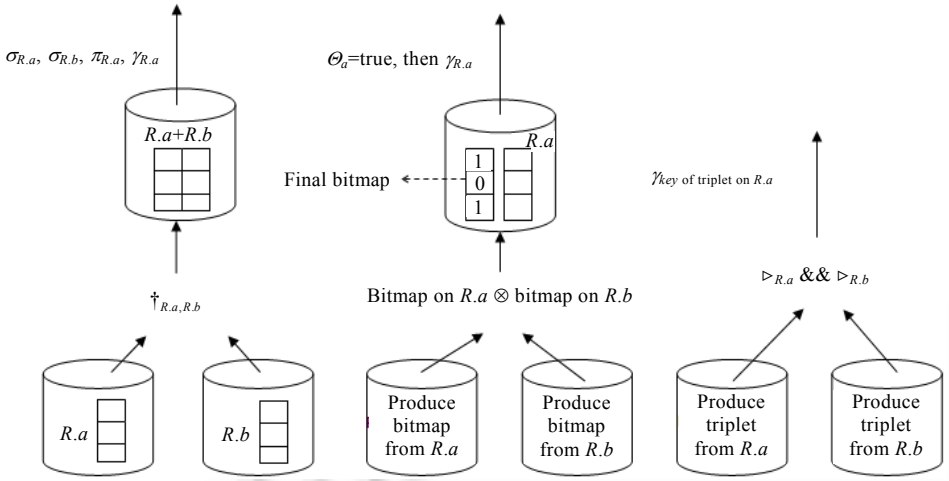
†:列存储中列拼接操作,$\dagger_{R.a,R.b}$ 拼接关系 R 的列 a 与列 b;

$\triangleright$:扫描三元组操作,如果 *key* 不为 0,返回 true.$\triangleright_a$ 扫描列 a 上产生的三元组;

Θ:扫描位图操作,对于位置位图为 1,返回 true.Θ_a 扫描列 a 上的位图.

列存储是对数据库物理层面的修改,在逻辑层和视图上,它等同于行存储.涉及到数据库的应用程序,无论数据库采用行存储还是列存储,都将接口视为面向行的接口.因此,列存储数据库必须在查询计划的某个点将逻辑上属于同一记录的各个属性值粘连起来,这个过程就叫作物化,它是投影的逆操作.根据物化的时机,可将物化操作分为两类:早物化(early materialization,简称 EM)和延迟物化(late materialization,简称 LM).每当在操作中需要某个列值时,这个列值就会被读入 CPU,并且添加在元组的中间表示上,这种物化方式称为早物化.由于早物化会导致 CPU 执行一些无用的工作,所以它并不总是最有效的[17].考虑以下场景:一个列存储数据库将一个表的各个列保存在单独的数据文件中,它们按相同的方式排序.假如一个查询在列 $R.a$ 和 $R.b$ 上有选择操作σ_1和σ_2(σ_1 有较高的选择率)以及在 $R.a$ 上有 GROUP BY 操作γ.在早物化的策略下,如图 4(a)所示,数据库执行的操作一般如下所示:读入列 $R.a$,$R.b$ 的数据块,合并列 $R.a$ 与 $R.b$($\dagger_{R.a,R.b}$).然后,将选择操作符σ_1和σ_2依次作用在这些元组上.符合过滤条件的元组进行投影操作π,最后进行分组聚集操作γ.不难看出,合并不满足过滤条件的属性是没有必要的操作.当选择率较低的时候,合并属性的大部分操作是无效的.处理该查询的一个更为高效的操作是保留两个中间位置列表;扫描列 $R.a$ 和 $R.b$ 的数据块,同时判断它们是否满足过滤条件σ_1和σ_2.对于满足过滤条件的元组,则在其位置列表相同的位置上标记为 1,否则为 0.之后,对两个位置列表进行“与”操作($\otimes$)形成最终位图.然后扫描最终位图(Θ),决定是否获取相应位置上的元组进行分组聚集操作γ.该操作过程如图 4(b)所示.随着硬件技术的发展,内存数据库的瓶颈所在已经从磁盘数据库的内存-磁盘转移到 CPU-内存上.CDDTA-

MMDB 得益于三元组的设计,可以进一步减少物化中内存访问和维护中间数据结构所带来的开销,我们称其为轻量级物化.轻量级物化与延迟物化的理念一致,即将物化操作尽可能地推迟,但是轻量级物化将过滤信息编码加入三元组,使得在存在过滤条件的情况下系统无需生成过滤位图,然后根据位图决定是否再次访问内存.如图4(c)所示,在轻量级物化的策略下,为了处理查询 Q,系统只需扫描三元组$\triangleright$,符合条件的 *key* 直接以流水线方式输送到分组聚集操作符π.



(a) 早物化 (b) 延迟物化 (c) 轻量级物化

Fig.4 The three kinds of materialization strategy

图 4 3 种物化策略

2.4 查询执行器

OLAP 上的查询分析操作一般表现为在事实表和维表之间连接操作的基础上对结果集上进行分组聚集等操作,因此,表连接和分组聚集操作是 ROLAP(relational OLAP)性能的两个重要决定因素.我们着重优化查询执行中的表连接算法,本文提出的查询执行算法能够以一趟扫描完成表连接、过滤与聚集等操作.本节给出朴素的轻量级物化下的表连接算法——LWMJoin(light-weight materialization join)及其优化.

2.4.1 朴素 LWMJoin 算法

LWMJoin 算法是 invisible join[16]的改进.invisible join 是针对星型模型的数据仓库而提出来的表连接算法,它分为 3 个阶段:

首先,将维表上的谓词条件作用在相应维表上,抽取符合过滤条件的元组的关键字构建哈希表;

然后,扫描事实表,并利用事实表外键探测维表上的哈希表,这一阶段确定了事实表上的过滤位图;

最后,根据过滤位图确定是否访问事实表相应位置上的元组构建结果集.如果过滤位图第 i 个位置上的值为 1,则需要检索事实表的第 i 条记录;否则,应舍弃事实表的第 i 条记录.

根据以上分析,invisible join 所耗费时间可以表达为

$$t_0+(2+1/\varepsilon)t_1,$$

其中,t_0为构建哈希表的时间,t_1为扫描事实表的时间($2t_1$的时间是:扫描事实表ε个外键属性列产生ε个过滤位图,扫描ε个过滤位图生成最终的过滤位图,由于过滤位图的大小(指的不是存储规模,而是长度)与事实表相同,因此可以近似认为这两趟扫描所需时间相同),t_2为扫描最终位图生成结果集所需时间.

扫描最终过滤位图所需时间约为 t_1/ε,因此,invisible join 所耗费的时间也可以用以下式子表达:

$$t_0+(2+1/\varepsilon)t_1.$$

与 invisible join 相比,LWMJoin 利用第 1 阶段产生的三元组,可以避免扫描事实表 2 次.以下是算法过程:

- 首先,维表上的子查询产生三元组⟨*surrogate*,*key*,*value*⟩,如果子查询只有过滤作用,例如 *RQ*1,那么产生过滤位图即可;
- 然后,顺序扫描事实表,利用事实表外键映射函数定位相应维表的三元组.如果这些三元组中的 *key* 值存在为 false,那么该条元组应被舍弃;否则,将 *key* 值与事实表度量属性拼接成一条元组,并将该元组以流水线的方式输送到下一个操作.

LWMJoin 避免谓词判断以后又需要再次访问内存,在 invisible join 的基础上进一步减少随机内存访问(随机内存访问会造成高速缓存命中缺失,内存数据库应当尽量避免此类操作).图 5 演示了 CDDTA-MMDB 处理查询 Q4.1(如图 3 所示)的步骤:首先,将查询分解并改写为 *RQ*1,*RQ*2,*RQ*3 和 *RQ*4 这 4 个子查询;之后,在相应维表上产生三元组或者是过滤位图(虚线的 *surrogate* 数组不作实际存储).第 2 阶段扫描事实表获取外键探测 *key* 数组,图 5 中虚线框所示为 Q4.1 中的分组器.对于满足过滤条件的元组,直接将分组器中相应的值以流水线的方式传送给下一个操作符.在这个例子中,事实表有 4 条元组(位置 1、位置 3、位置 5、位置 6)满足条件,结果集有两组.LWMJoin 所耗费的时间由产生三元组的 t_0(与 invisible join 第 1 阶段时间相同)和一趟扫描事实表时间 t_1 组成,因此, LWMJoin 所耗费时间大约是 invisible join 的 2+1/ε之一.由于轻量级物化的结果,LWMJoin 实际所耗时间比 invisible join 在理论上分析的还要少,后面的实验验证了这一点.图 6 给出了算法流程图步骤:首先初始化变量,将游标置于事实表第 1 条元组.根据 SQL 解析的信息,获取查询相关外键值探测 *key vector*.如果某个 *key* 值不符合条件,则转而处理下一条元组;如果根据事实表外键所访问的三元组 *key* 值都大于 0,则构造物理元组并将元组以流水线方式输送到下一操作符.

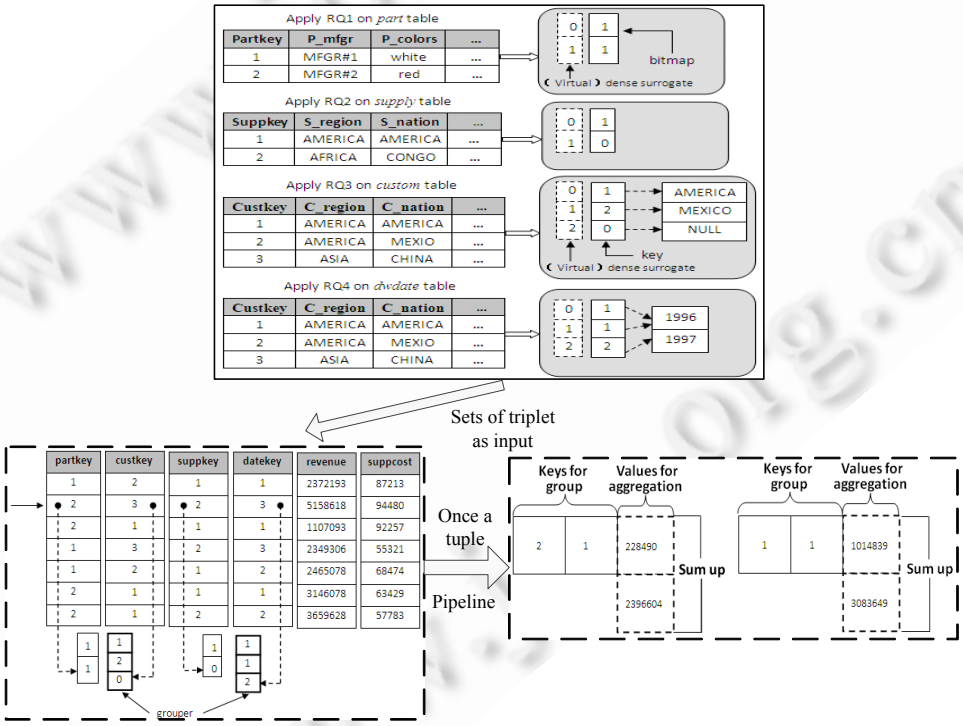


Fig.5 Query execution process example

图 5 查询执行流程示例

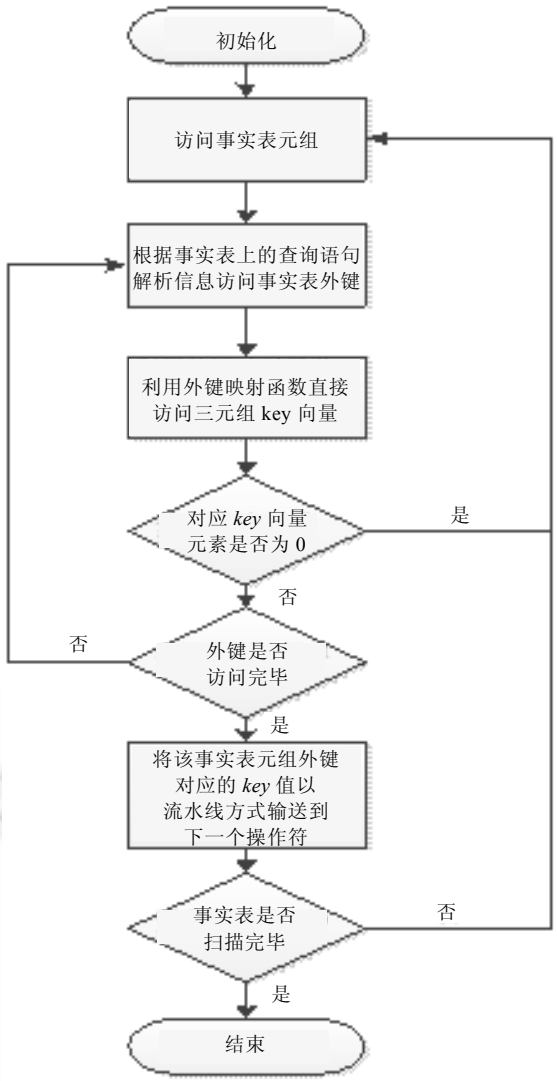


Fig.6 The naive LWMJoin algorithm flow chart

图 6 朴素 LWMJoin 算法流程图

2.4.2 查询优化

事实上,选择外键探测 *key vector* 的顺序是很重要的.如果事实表上有过滤条件,也应当把这个因素考虑进来.我们倾向于优先探测具有最低选择率的 *key vector*,依次按选择率执行谓词过滤判断.这个优化的原则是很简单的:尽量减少内存访问.一旦某个 *key* 的值为 false,即可停止访问内存获取 *key vector* 元素的操作,转而执行下一跳记录.我们通过建立统计直方图,利用 selinger 风格来评估选择率.例如,有 3 个维表(t_1,t_2,t_3),3 个等值选择操作$\sigma_{A=a}$,$\sigma_{B=b}$和$\sigma_{C=c}$分别作用在这 3 个维表上.如果下面的关系成立:

$$\frac{n_3'}{n_3}<\frac{n_1'}{n_1}<\frac{n_2'}{n_2} \tag{1}$$

其中,n_x'是维表 t_x 中满足条件的元组数,n_x是维表大小.如果公式(1)成立,那么首先访问维表 t_3 的 *key vector*,然后依次是 t_1 和 t_2.在查询分解过程中,每个维表上的选择率可以在生成 triplet 时进行精确的计算,只需对相关维表

的选择率进行简单排序,调整事实表过滤顺序即可,对查询处理的执行计划没有其他影响.

3 实验结果与分析

CDDTA-MMDB 是一个内存列存储数据库查询执行引擎,是针对星型模型数据仓库上的 OLAP 而提出的解决方案.我们将采用星型模型测试基准[25,26](star schema benchmark,简称 SSB)来评估执行引擎的性能.SSB 是标准化的 TPC-H,与 TPC-H 不同的是,SSB 只有一个事实表 LINEORDER.这个事实表是通过合并 TPC-H 的 LINEITEM 和 ORDERS 表而得到的,有 17 个属性.事实表记录个人的订单信息.LINEORDER 的主键是组合键,包含 ORDERKEY 和 LINENUMBER 属性.事实表的其他属性包含指向 CUSTOMER,PART,SUPPLIER 和 DATE 表的 4 个外键以及订单的其他信息,例如优先级、价格、折扣和发货日期等.SSB 包含 4 个维表:CUSTOMER,PART,SUPPLIER 和 DATE 表.如同 TPC-H,SSB 也有一个扩展因子(scale factor),用来定义测试基准集的大小.各个表的大小都是相对这个扩展因子来确定的.本文的实验中使用的扩展因子为 1,10,50 以及 100.此外,SSB 有 13 个查询,分为 4 组.更多关于 SSB 的细节可参考文献[25,26].

加载到内存的数据包括事实表中与实验相关的属性、所有的维表.实验环境如下:48GB 的内存、两路 Intel® xeon® E5645 处理器 2.4GHZ(每个处理器有 6 个超标量的处理核心,每个核心支持 2 个并发的物理线程)、1TB 磁盘空间、64 位 Ubuntu 10.10 操作系统.我们的关注点是轻量级物化所带来的性能优化、CDDTA-MMDB 的整体性能以及系统在多核环境下的扩展性.

3.1 轻量级物化与延迟物化比较

这个实验中扩展因子 *SF*=100,忽略没有 GROUP BY 操作的查询 Q1.*x*.我们模拟延迟物化根据最终位图访问维表操作,强制性地让 LWMJoin 在谓词判断以后访问维表,以获取数据进行聚集操作(称为 CDDTA-LMJoin,late materialization join).同时,测试轻量级物化下这个阶段所耗费的时间.令我们感到意外的是 Q3.1,如图 7 所示,Q3.1 中分组聚集所用时间占据了总时间的 61%,超过了事实表扫描的时间.Q4.1,Q2.1 和 Q4.2 中分组聚集所用时间占总时间的百分比分别是 35%,21%和 17%.相比之下,在上述 4 个查询中,LWMJoin 已经将这个阶段耗费的时间所占百分比下降到 28%,13%,8%和 5%.在参与实验的其余查询中,LWMJoin 在扫描事实表后的阶段所耗费时间基本上也比 CDDTA-LMJoin 要少.这个差别就在于扫描事实表以后,如果该条元组满足过滤条件,则 CDDTA-LMJoin 需要再次访问内存获取数据以进行下一步操作,而这种随机内存访问模式容易造成高速缓存命中缺失.所以,CDDTA-LMJion 所耗费的时间比采用轻量级物化的 LWMJoin 时间要长,尤其是当选择率较高的情况下,例如 Q2.1,Q3.1,Q4.1.

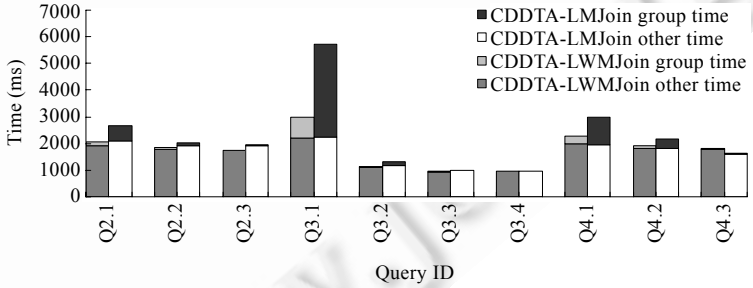


Fig.7 CDDTA-LWMJoin and CDDTA-LMJoin time-consuming comparison in aggregation and grouping

图 7 CDDTA-LWMJoin 与 CDDTA-LMJoin 在分组聚集时间所耗费时间对比

3.2 CDDTA-MMDB总体性能评估

在这一节中,我们测试系统 CDDTA-MMDB 的总体性能.在不同的数据量下(*SF* 分别为 1,10,50,100),将系统与 MonetDB 以及 C-Store 的 invisible join 算法进行比较.CDDTA-MMDB 与 invisible join 能够检测系统配置,

将事实表根据 CPU 参数(支持的物理线程数)水平均匀划分,然后将划分后的表赋予各个物理线程.在这个实验中使用的 MonetDB 是支持多核多线程环境下的版本.我们还比较了朴素 LWJoin 与过滤顺序优化的 LWMJoin.图 8 给出了 *SF*=100 的情况下,朴素的 LWMJoin 以及带过滤顺序优化的 LWMJoin 的性能对比.查询 Q1.*x* 有两个过滤条件:一个作用在维表 dwdate 上,另一个作用在事实表上.如果先判断事实表上的谓词条件,那么系统所用查询时间会大幅度提升.在其余的查询中,带优化规则的系统也有较好的性能.图 9 是在 *SF*=1,10,50,100 的情况下,CDDTA-MMDB 与 MonetDB(v11.7.9)以及 invisible join 的比较.

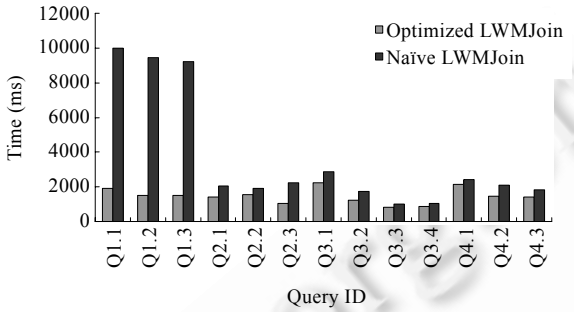


Fig.8 Naive LWMJoin and optimized LWMJoin with filter order

图 8 朴素 LWMJoin 与过滤顺序优化规则的 LWMJoin

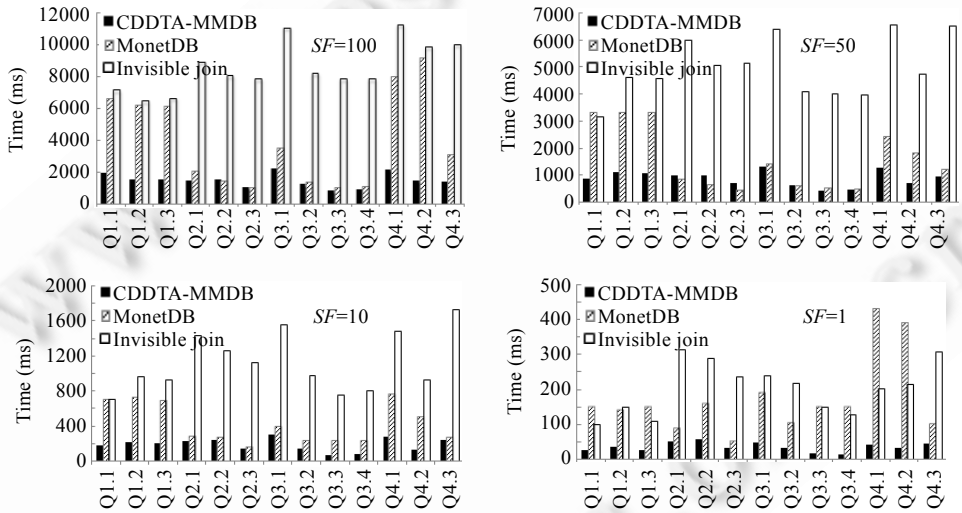


Fig.9 CDDTA-MMDB vs. MonetDB vs. Invisible join time-consuming under different SF

图 9 不同 SF 下,CDDTA-MMDB vs. MonetDB vs. invisible join 所耗费时间对比

我们让进行比较的系统在数据集上重复运行若干次,使得实验是在“热”数据上进行,以消除 I/O 影响.在这个实验中,CDDTA-MMDB 具有最优性能.总体而言,C-Store 的 invisible join 比 CDDTA-MMDB 耗费的时间多 5 倍左右,这基本符合第 2.4.1 节的算法分析.在不同的 *SF* 下,MonetDB 处理 Q2.*x*,Q3.*x* 以及 Q4.3 所需时间基本上比其他查询要少,这主要是因为查询 Q2.*x* 以及 Q3.*x* 需要处理的维表数据较小以及 Q4.3 的选择率很低.MonetDB 的基数聚集表连接算法(radix cluster join)是高速缓存敏感的.当数据集能够全部放入高速缓存时,MonetDB 具有如下很好的性能:

1) *SF*=50,100.CDDTA-MMDB 与 MonetDB 在处理 Q2.*x*,Q3.*x* 所需时间相差不大,MonetDB 甚至要略优于我们的系统;

2) *SF*=1,10.*key vector* 能够全部放入高速缓存,我们的查询处理算法的简单性占据了优势,从而在这几个查询上,CDDTA-MMDB 的性能要全部优于 MonetDB;
3) *SF*=1,10,50,100.在其余的查询中(Q1.*x*,Q4.*x*),我们的系统都要明显优于 MonetDB.在 *SF*=100 的情况下,CDDTA-MMDB 处理实验中的 13 查询所用总时间为 19 920ms,平均每个查询耗时 1.5s 左右,而 MonetDB 为 50 311ms,平均每个查询耗时约 3.8s.

3.3 CDDTA-MMDB扩展性

我们的另外一个关注点是系统在多核平台上的扩展性,将处理线程个数从 1 增加到 12,测试 CDDTA-MMDB 的加速比.选取 Q1.1,Q2.1,Q3.1 和 Q4.1 进行实验.纵轴是一个线程执行任务的时间除以 *n* 个线程处理任务的时间的比率.如图 10 所示,CDDTA-MMDB 基本上接近线性加速比.

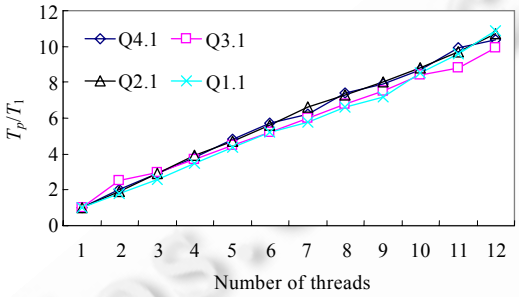


Fig.10 CDDTA-MMDB scalability on multi-core platform

图 10 CDDTA-MMDB 在多核平台上的扩展性

4 总 结

本文的主要贡献在于在新硬件平台下,针对大规模数据的 OLAP 查询所开发的一个新的查询执行引擎 CDDTA-MMDB.这个查询执行引擎引进了一种新的表连接算法——LWMJoin,该算法能够消除传统的数据库中多表连接的性能瓶颈.另外,我们在 CDDTA-MMDB 中使用了一种新的物化策略——轻量级物化.轻量级物化具有延迟物化的优点,能够避免 CPU 产生额外的无用工作.与延迟物化相比,它又能减少内存访问和中间数据结构的产生.实验结果表明,CDDTA-MMDB 是高效的,且具有较好的扩展性.这个查询引擎部署在集群系统中,利用反转星型模型[8]的处理模式,可以轻松地应对 TB 级甚至是 PB 级的海量数据分析.未来的工作是把轻量级物化策略加入到开源的数据库系统,例如 MonetDB,以及在 CDDTA-MMDB 中加入并发控制.

References:

[1] Douglas L. The Importance of 'Big Data': A Definition. Gartner, G00235055, 2012.

[2] Boncz P, Zukowski M, Nes N. MonetDB/X100: Hyper-Pipelining query execution. In: Ailamaki A, ed. Proc. of the CIDR 2005. Asilomar, 2005. 225−237.

[3] Boncz P, Grust T, van Keulen M, Manegold S, Rittinger J, Teubner J. MonetDB/XQuery: A fast xquery processor powered by a relational engine. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Chicago: ACM Press, 2006. 479−490. [doi: 10.1145/1142473.1142527]

[4] Larson PA, Hanson EN, Price SL. Columnar storage in SQL server 2012. In: Kementsietsidis A, Vaz Salles MN, eds. Proc. of the IEEE 28th Int'l Conf. on Data Engineering (ICDE 2012).Washington: IEEE Computer Society, 2012. 15−20.

[5] Boncz PA, Kersten ML. MIL primitives for querying a fragmented world. In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, eds. Proc. of the 25th Int'l Conf. on Very Large Data Bases (VLDB'99). Edinburgh: Morgan Kaufmann Publishers, 1999. 101−119.

[6] Plattner H. A common database approach for OLTP and OLAP using an in-memory column database. In: Çetintemel U, Zdonik SB, Kossmann D, Tatbul N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2009). Rhode Island: ACM Press, 2009. 1−2. [doi: 10.1145/1559845.1559846]

[7] Sikka V, Färber F, Lehner W, Cha SK, Peh T, Bornhövd C. Efficient transaction processing in SAP HANA database—The end of a column store myth. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2012). Scottsdale: ACM Press, 2012. 731−741. [doi: 10.1145/2213836.2213946]

[8] Zhang YS, Jiao M, Wang ZW, Wang S, Zhou X. One-Size-Fits-All OLAP technique for big data analysis. Chinese Journal of Computers, 2011,34(10):1936−1946 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01936]

[9] Wang HJ, Qin XP, Zhang YS, Wang S, Wang ZW. LinearDB: A relational approach to make data warehouse scale like MapReduce. In: Yu JX, Kim MH, Unland R, eds. Proc. of the 16th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2011). Lecture Notes in Computer Science, Hong Kong, 2011. 306−320. [doi: 10.1007/978-3-642-20152-3_23]

[10] Wang S, Wang HJ, Qin XP, Zhou X. Architecting big data: Challenges, studies and forecasts. Chinese Journal of Computers, 2011, 34(10):1742−1752 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01741]

[11] Copeland GP, Khoshaian S. A decomposition storage mode. In: Navathe SB, ed. Proc. of the 1985 ACM SIGMOD Int'l Conf. on Management of Data. Austin: ACM Press, 1985. 268−279. [doi: 10.1145/318898.318923]

[12] Boncz P, Kersten ML, Manegold S. Breaking the memory wall in MonetDB. Communications of the ACM, 2008,51:77−85. [doi: 10.1145/1409360.1409380]

[13] Stonebraker M, Abadi DJ, Batkin A, Chen XD, Cherniack M, Ferreira M, Lau E, Lin A, Madden S, O'Neil E, O'Neil P, Rasin A, Tran T, Zdonik S. C-Store: A column-oriented DBMS. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson PA, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. Trondheim: ACM Press, 2005. 553−564.

[14] Selinger PG, Astrahan MM, Chamberlin DD, Lorie RA, Price TG. Access path selection in a relational database. In: Bernstein PA, ed. Proc. of the 1979 ACM SIGMOD Int'l Conf. on Management of Data. Boston: ACM Press, 1979. 23−34.

[15] Hong W, Stonebraker M. Exploiting InteroperatorParallelism in XPRS. In: Stonebraker M, ed. Proc. of the 1992 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 1992. 19−28.

[16] Abadi DJ, Madden SR, Hachem N. Column-Stores vs. row-stores: How different are they really. In: Wang JTL, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2008). Vancouver: ACM Press, 2008. 967−980. [doi: 10.1145/1376616.1376712]

[17] Abadi DJ, Myers DS, DeWitt DJ, Madden SR. Materialization strategies in a column-oriented DBMS. In: Chirkova R, Dogac A, Özsu MT, Sellis TK, eds. Proc. of the 23rd Int'l Conf. on Data Engineering (ICDE 2007). Istanbul: IEEE, 2007. 466−475. [doi: 10.1109/ICDE.2007.367892]

[18] Halverson A, Beckmann J, Naughton J. A comparisonof C-store and row-store in a common framework. Technical Report, TR1566, UW Madison Department of CS, 2006.

[19] Harizopoulos S, Liang V, Abadi DJ, Madden S. Performance tradeoffs in read-optimized databases. In: Dayal U, Whang KY, Lomet DB, Alonso G, Lohman GM, Kersten ML, Cha SK, Kim YK, eds. Proc. of the 32nd Int'l Conf. on Very Large Data Bases. Seoul: ACM Press, 2006. 487−498.

[20] Harizopoulos S, Abadi D, Madden S, Stonebraker M. OLTP through the looking glass, and what we found there. In: Wang JTL, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2008). ACM Press, 2008. 981−992. [doi: 10.1145/1376616.1376713]

[21] DeBrabant J, Pavlo A, Tu S, Stonebraker M, Zdonik S. Anti-Caching: A new approach to database management system architecture. Proc. of the VLDB Endowment, 2013,6(14):1942−1953.

[22] Youssefiand K, Wong E. Query processing in a relational database Management system. In: Furtado AL, Morgan HL, eds. Proc. of the 5th Int'l Conf. on Very Large Data Bases. Rio de Janeiro: IEEE, 1979. 409−417.

[23] Held GD, Stonebraker M, Wong E. INGRES—A relational data base management system. In: Proc. of the 1975 National Computer Conf. on American Federation of Information Processing Societies. Anaheim: AFIPS Press, 1975. 409−417. http://dl.acm.org/citation.cfm?doid=1499949.1500029

[24] Garcia-Molina H, Ullman JD, Widom J. Database System Implementation. Upper Saddle River: Prentice Hall, 2000.

[25] O'Neil PE, O'Neil EJ, Chen X. The star schema benchmark (SSB). 2009. http://www.cs.umb.edu/~poneil/StarSchemaB.PDF

[26] O'Neil PE, Chen X, O'Neil EJ. Adjoined dimension column index to improve star schema query performance. In: Alonso G, Blakeley JA, Chen ALP, eds. Proc. of the 24th Int'l Conf. on Data Engineering (ICDE 2008). Cancún: IEEE, 2008. 1409−1411.

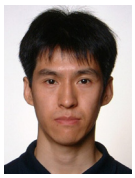
附中文参考文献:

[8] 张延松,焦敏,王占伟,王珊,周烜.海量数据分析的 One-size-fits-all OLAP 技术.计算机学报,2011,34(10):1936−1946. [doi: 10.3724/SP.J.1016.2011.01936]

[10] 王珊,王会举,覃雄派,周烜.架构大数据:挑战、现状与展望.计算机学报,2011,34(10):1742−1752. [doi: 10.3724/SP.J.1016.2011.01741]

朱阅岸(1983－),男,广东梅州人,博士生,主要研究领域为高性能数据库,信息检索.
E-mail: iwillgoon@ruc.edu.cn

周烜(1979－),男,博士,副教授,主要研究领域为高性能数据,信息检索.
E-mail: zhou.xuan@outlook.com



张延松(1973－),男,博士,讲师,主要研究领域为内存数据库,OLAP.
E-mail: Zhangys_ruc@hotmail.com

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,内存数据库,非结构化数据库,数据仓库与数据分析.
E-mail: swang@ruc.edu.cn