

面向 ROLAP 的时态垂直粗分区方法

李文海^{1),2)} 冯玉才²⁾ 马晓鸣^{3),4)} 付 铨²⁾ 胡文斌¹⁾

¹⁾(武汉大学计算机学院计算机工程系 武汉 430079)

²⁾(华中科技大学计算机科学与技术学院数据库与多媒体技术研究所 武汉 430074)

³⁾(武汉大学经济与管理学院 武汉 430072)

⁴⁾(湖北省国家税务局 武汉 430071)

摘 要 数据仓库为海量数据上的决策支持提供了一个高效的信息管理平台,ROLAP 利用关系型数据仓库操纵灵活和技术成熟等优势,为面向数据仓库的分析和决策提供了有效的存取、建模和操作方法.然而,传统关系存取方法造成 ROLAP 的 I/O 有效性面临严峻的挑战.首先通过分析 DSS 应用的特点,提出了关系算子访问基表属性的时态行为,定义了算子对属性的时态局部访问.通过对查询样本集的解析建立算子与属性的时态访问映射矩阵,将有效增益作为属性的聚类准则得到时态访问模型 PD.最后,给出了求解该模型的粗集算法以及依据聚类结果设计的基表属性的垂直分区方案.实验证明:在决策支持应用中,该方法的效率优于同类的其它优化分区方法.

关键词 关系算子;时态聚类;关系存储;不可辨识度;粗集模型

中图法分类号 TP18

ROLAP-Oriented Temporal Vertical Partitioning Method Based on Rough Sets

LI Wen-Hai^{1),2)} FENG Yu-Cai²⁾ MA Xiao-Ming^{3),4)} FU Quan²⁾ HU Wen-Bin¹⁾

¹⁾(Department of Computer Engineering, Computer School, Wuhan University, Wuhan 430079)

²⁾(Institute of Database and Multimedia Technology, College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

³⁾(Economics and Management School of Wuhan University, Wuhan 430072)

⁴⁾(Hubei State Administration of Taxation, Wuhan 430071)

Abstract As the essential element of DSS applications, data warehouse coupled with ROLAP provides an administrant platform with high performance. Especially, it exhibits great efficiency in model construction, data access and operation. Conventional access method of the relational data warehouse is, however, faced with new challenges due to the poor I/O efficiency in ROLAP. In this paper, the characters of DSS are analyzed and the temporal-locality attributes accessed by the operators are proposed in terms of the temporal operations. After this, the corresponding mapping matrix of the temporal access is established by parsing the query template set. With the rough set model, the model PD takes the relational attributes for the clustering objects and regards the queries as the vector attributes. With the proposition of rough clustering algorithm, a series of partitions are obtained through a measurement named effectual benefit criteria under different indiscernibility thresholds. At last, the partitions are evaluated and the storage scheme derived from the most effective partition is designed for the vertically partitioning sub-tables of the relations. Experiments show the proposed scheme is superior to other optimization strategies.

收稿日期:2006-11-02;最终修改稿收到日期:2008-05-05. 本课题得到国家“八六三”高技术研究发展计划项目基金(2004AA4Z3020, 2005AA4Z3030)、科技部电子政务关键技术及应用系统研究项目(2001BA110B01)与湖北省自然科学基金(2006ABA218)资助. 李文海,男,1979年生,博士,讲师,主要研究方向为关系数据库优化、知识获取理论及应用. E-mail: lwhaymail@21cn.com. 冯玉才,男,1945年生,教授,博士生导师,主要研究领域为关系数据库理论、数据库管理系统和多媒体技术的理论研究与实现. 马晓鸣,女,1980年生,博士研究生,主要研究方向为区域经济的数量分析及其决策问题. 付 铨,男,1977年生,博士研究生,主要研究方向为关系数据库安全与实现. 胡文斌,男,1977年生,副教授,主要研究方向为决策支持与计算机仿真.

Keywords relational operator; temporal clustering; relational storage; indiscernibility degree; rough set model

1 引言

通过对异构交易数据库的数据抽取和集成,数据仓库用于保存跨全局、海量的历史数据,并涵盖一系列有益于领域专家准确、迅速做出决定的决策支持(DSS)技术^[1].联机分析(OLAP)技术通过对仓库数据执行复杂的查询分析,创造性地获取有关主题趋势的预测信息.在典型的数据仓库中,数据多采用事实表-维表的星型方式建模,查询具有复杂、即席和大数据量等特点,且一般包含大量的连接和聚集^[1],因此快速的决策支持响应有赖于高效的查询处理.

关系型 OLAP(ROLAP)利用关系数据库操纵灵活和技术成熟等优势,为数据仓库的分析和决策支持提供了高效的信息建模、存取和操作方法.近年来,面向 ROLAP 的优化已成为关系数据库的一个重要研究方向,而作为 ROLAP 的基础,高效的存储策略亦被日益广泛地研究^[1-2].关系数据库存储优化依据所面向的硬件对象不同可分为对 I/O 优化^[2-7]和内存存储优化^[8-9].依据所面向功能对象的不同,I/O 优化可分为基本关系表(基表)分布优化和存取结构优化;依据优化策略的不同,基表分布优化又可分为水平分区和垂直分区优化^[5].I/O 优化与内存优化实质上都是利用冗余结构改变基表属性布局,以提高硬件的工作效率和系统吞吐率.不同之处在于:前者在外存储部件中分离基表元组,一般将访问基表的 I/O 增益与冗余操作的代价之差作为分区方案的衡量标准;后者通过改变了已有的 I/O 代价估算方法,将深层次内存的存取代价作为系统设计的依据.

DSS 应用侧重于获取仓库数据中蕴含的决策信息,因而造成 ROLAP 服务器与传统事务处理型服务器的设计需求不同.ROLAP 中关系事实表具有高维和大数据量的特点^[1].这使得处理即席查询时,较大的关系表只能部分读入内存,进而引起 I/O 操作的颠簸^[1,5].与 OLTP 系统不同,OLAP 系统保存的是历史数据,一般不对基表的逻辑模式及其元组做删除和修改,用户主要关心 OLAP 系统的查询性能;基表的逻辑模式仅反映某一特定现实活动和实体的各种数量特征,查询局限于部分与决策密切

相关的数量信息,且这种相关性相对固定.已有属性垂直分区策略以查询事务为最小的分析粒度,以查询访问属性的统计值为计算依据,对分区所有可能方案计算查询代价.这种求取最小查询代价的方法是 NP 难的^[9].

应当注意到,关系数据库中物理算子(physical operator,下称算子)的执行次序遵循一定原则^[10],且部分算子在执行期间存在结果依赖性^[11-12],算子执行的次序特征使其对属性的访问具有时间上的局部特性.当物理内存无法一次满足数据请求时,发掘算子的时态局部性对提高 I/O 有效性具有重要的意义.据此,本文提出一种基于粗集模型的关系数据垂直分区方法.利用 DSS 查询样本的最优执行计划,将(基表)属性作为对象进行时态分析,将查询样本作为对象的特征属性(attributes,下称特征),将访问属性的有效 I/O 增益作为对象之间的聚类距离,通过粗集聚类获取在操作上具有紧密时间局部性的属性垂直分区.基于 TPC-H 基准和一个真实的税收数据仓库实例,本文以 DM4 关系数据库管理系统作为实验平台,对时态分区方法与已有方法进行实验比较和分析论证.

2 相关研究

垂直分区优化在分区方案的设计上先后出现了 3 种不同的方法:调用数据库估算器优化、完全属性分区和代价模型优化.

文献[2-3]根据全体查询样本的访问统计对属性进行初始合并. Autopart^[2]调用数据库的代价估算器对初始合并的二次组合进行贪婪估算,进而选取代价最低的作为最终分区方案.为降低贪婪估算的时间复杂度,GA^[3]采用基因算法获取具有近似最低估算代价的分区方案.前者具有较高时间复杂度,但其分区方案具有更高的 I/O 效率^[3].

Fractured Mirrors^[4]为每个基表存储冗余副本,该副本按单一属性进行拆分存储,并为每个子表设计相应元组标识以便提高子表的连接效率.

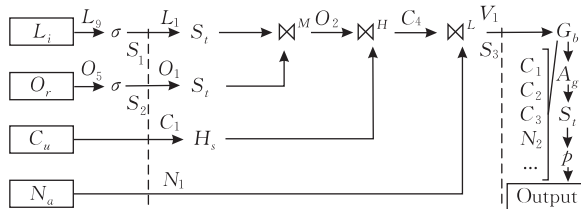
文献[5-7]同样首先获取查询样本的初始合并,然后通过构造代价估算公式获取最优二次组合.其中,VPC^[5]针对所有可能的二次组合子表,计算查询样本的扫描和连接的代价差异以确定最优方案;

CSP^[6] 和 OBP^[7] 为所有 I/O 相关的关系算子设计代价模型, 取二次组合方案中代价最小者作为最终分区方案. 由于 VPC 将代价因素限制在分区前后的效率增益和耗损的差值上, 摒弃了代价模型中其它因素对代价估算精确度的影响, 故其对分区性能的度量更为精确, 且较 Autopart 具有更快的执行效率和通用性.

文献[12]采取算子级的负载分析获取初始分区方案, 然后通过代价估算求解基表的最优水平分布, 以获取磁盘组的最大并行 I/O 吞吐率. 对于 ROLAP 的垂直分区问题, 查询统计划分和全局代价模型很难充分发掘分区数据的访问有效性. 利用算子级负载分析和增益分析技术可以更加充分地发掘查询样本中算子访问基表属性的时态行为, 通过合理的垂直分区方案提高查询性能.

3 时态访问聚类

关系优化引擎针对给定查询和基表, 根据系统维护的参数统计信息, 搜索待选计划空间中代价最低的作为执行计划. 待选计划空间实质上是选择 σ , 投影 π 和连接 \bowtie 等逻辑运算符所对应的物理算子的符合固定约束的序列集^[10], 多数 RDBMS 使用计划树表达并选取查询计划. 本研究基于国产 DM4 数据库, 其通过“左深度树”^[10] 表达并估算执行计划: 以中间结点表示算子, 以叶结点表示基表, 基表属性及代价参数以中间结点的参数形式给出. 以 TPC 基准中查询 10^① 为例, 图 1 列出该查询在 DM4 中的最优计划.



RI/QO: 关系输入/查询输出 L_i, N_i, C_i, O_i : 关系属性
 L_i, O_r, C_u, N_a : 基准关系 Lineitem, Orders, Customer, Nation
 S_i : 元组势 $\bowtie^L, \bowtie^M, \bowtie^H$: 迭代, 归并, 哈希连接
 σ, p : 选择, 投影 G_b, S_i, H_s, A_g : 分组, 排序, 哈希, 聚集

图 1 Query10 的最优查询计划示意图(左侧矩形标识查询涉及的 4 个基表, 线段上的参数标识算子操作的基表(临时表)属性和预计元组条目, 属性以其所在基表定义中的序列号标识, 箭头标明算子与属性的对应关系, 虚线标识时态局部划分)

对于 I/O 密集型的 DSS 应用, 影响查询 I/O 数量的主要因素有请求元组数量、内存容量、算子执行模式. 传统 RDBMS 采取 N-ary 存储模型^[10] (NSM) 组织基表数据, 用元组构成磁盘块进行 I/O 交换. 而另一方面, 基于一定的优化原则和算子间的结果依赖性, 查询优化器使用“算子下推”和“blocking”等策略实现算子(集)的优先级, 以保证高优先级算子的数据请求首先得到满足(如虚线所示). 这种交换与访问的粒度差异会导致两类无效数据被频繁地调入与调出, 如图 1 所示.

(1) 查询并不涉及的属性: 基表 L_i 包含 16 个属性, 但仅有 4 个属性 L_9, L_1, L_6 和 L_7 被访问;

(2) 近期并不使用的属性: (1) 中 4 个属性在执行时并非同时被请求, 而是以 $L_9, L_1, \langle L_6, L_7 \rangle$ 为序由 3 个算子 σ, \bowtie^M, A_g 分别在不同时间段操作.

上述两类无效数据在处理高维数据时造成属性数据的颠簸^[12], 极大降低了 I/O 效率. 对于给定查询 Q 的“左深度”计划树 $LTree(Q)$, 我们定义算子访问的时态局部性刻画算子对基表 R 中属性的局部性请求. 令查询 Q 的算子在集合 $\{\sigma, \pi, \bowtie, \psi\}$ 中取值 (ψ 为除选择、投影和连接以外的算子), 基表 $R = \{A_1 A_2 \cdots A_n\}$ 由 n 个属性组成. 基于对 $LTree(Q)$ 的后序遍历, 不妨设树中任意算子结点 $O_i \in LTree(Q)$ 在时刻 t_i 开始执行.

定义 1. 称算子 $O_i \in LTree(Q)$ 为阻塞算子, 如果在 t_i 前 $\forall j < i, O_j$ 的 I/O 请求已被满足, 记作 $blk(O_i)$.

执行计划中, 主要有两类算子以阻塞的方式先于后续算子完成 I/O 请求: 低代价算子被首先执行以减少高代价算子的数据势; 算子的执行结果被其它算子作为操作对象. 这些阻塞算子将 $LTree(Q)$ 的遍历序列划分为有限个时态局部集.

定义 2. 称 $O = \{O_i \cdots O_j\}$ 为一个时态局部算子集, 若 $\forall O_k \in O, t_i \leq t_k < t_j \wedge blk(O_{i-1}) \wedge blk(O_j) \wedge \neg blk(O_k)$.

进而, 若 $LTree(Q)$ 的所有时态局部算子集构成集合 $\mathcal{O} = \{O^1, O^2, \cdots, O^K\}$, 则 R 中被 $\forall O^k \in \mathcal{O}$ 访问的属性集构成一个时态访问聚类 R^k , 未被 \mathcal{O} 中任何算子集访问的属性集记作 $R^0, \mathcal{C} = \{R^0, R^1, \cdots, R^K\}$ 构成 R 在 \mathcal{O} 下的时态划分. 依据 \mathcal{C} 中聚类的相互关系可将 \mathcal{C} 分为两类.

① TPC BENCHMARK™ H standard specification (Decision support), Revision 2.3.0. <http://www.tpc.org/tpch>

定义 3. 若 \mathcal{C} 的所有聚类互不相交, 则称 \mathcal{C} 为时态分区且记作 C ; 否则称 \mathcal{C} 为时态覆盖且记作 U .

给定查询 Q 的“左深度”树 $LTree(Q)$, 利用后序遍历的子树分解算法 DPOS (Decomposition of Post Order Subtree), 我们得到系统中基表 R 的有序聚类及其相应参数的时态分区 C , 基本流程如下:

输入: 查询计划左深度树 $LTree(Q)$

输出: 时态聚类结果 C

算法流程:

1. 初始化有序集 \mathcal{S} , 聚类集 \mathcal{T} , 临时聚类 c 和结果集 C ;
2. 后序遍历 $LTree(Q)$ 生成

$$\mathcal{S} = \{s_i : \langle O_i, R_{j-}^-, S_k, V_l \cdot R_{j+}^+, S_{k'} \rangle\};$$
3. for ($\forall s_i \in \mathcal{S}; s_i \neq \text{NULL}; s_i = \text{Next}(\mathcal{S})$) do

// 依据 \mathcal{S} 生成算子集
4. $c \cup = s_i$; // 合并算子
5. if ($\text{blk}(s_i \cdot O_i)$) then // 依据算子执行模式划分算子集
6. $\mathcal{T} \cup = c; \text{new}(c)$; // 算子集加入非阻塞聚类集
7. end if
8. end for
9. for ($\forall c \in \mathcal{T}; I++$) do

// 生成 \mathcal{T} 中聚类对应的属性聚类集 C
10. $\text{new}(c^I : \langle R^I | O^I, T^I, L^I, V^I, S^I \rangle; \text{new}(T))$;
11. $R^0 = \bar{T}$; $R^I = \bigcup_{\forall s_i \in c} \{R_{j-} : s_i \cdot R_{j-}^- \in R\} \cup \{R_{j+} : s_i \cdot V_l \cdot R_{j+}^+ \in R\} - T$;
12. if ($s \cdot V_l = \text{NULL}$) then // 若 c 为单目聚类算子集
13. $s = s_i | \max_{\forall s_i \in c} \max(s_i \cdot S_k, s_i \cdot S_{k'})$ // $\max()$ 返回非空者
14. else // 若 c 为双目聚类算子集
15. $s = s_i | \max_{\forall s_i \in c} \min(s_i \cdot S_k, s_i \cdot S_{k'})$;
- // 选取规模最大的算子
16. $O^I = s \cdot O_i$; $L^I = s \cdot S_k$; $S^I = s \cdot S_{k'}$;
- // 度量算子集参数
17. end if
18. if ($R \subset s \cdot V_l$) then // 若 s 的输入包含 R 属性
19. $V^I = s \cdot V_l - \bar{T}$; $T^I = s \cdot R^-$;
20. else // 若 s 直接以 R 属性为操作对象
21. $V^I = s \cdot V_l$; $T^I = s \cdot R^- - \bar{T}$;
22. end if
23. $T = T \cup R^I$; // 属性加入临时类
24. $C \cup = c^I$; // 临时类加入结果集
25. end for

基于阻塞算子划分 $LTree(Q)$ 的遍历序列, 每个极小子序列内的算子集以非阻塞方式执行, 聚类 R^I 中属性被 O^I 所在的算子集交迭地访问, 且平均间隔时间远小于 O^I 的执行时间 (如流水线、缓冲等执行方式); 算子集之间以阻塞方式执行, 对两聚类 $R^I, R^J | J > I$ 访问开始时刻的间隔大于算子集 O^I 的执行周期. 利用 DM4 优化引擎对算子的参数化表达, DPOS 算法可以调用相应的关系表及其元组势.

行 1 用于初始化五元组的有序集 \mathcal{S} 、有序聚类集 \mathcal{T} 和一个临时聚类 c , 并初始化结果集 C ; 行 2~10 利用后序遍历生成有序集 \mathcal{S} . DM4 将 N 目算子转换为 $N-1$ 个双目算子, 故 $LTree(Q)$ 中操作 R 的算子与 \mathcal{S} 中五元组 $\langle O_i, R_{j-}^-, S_k, V_l \cdot R_{j+}^+, S_{k'} \rangle$ 一一对应. 其中, $R^+ = R \vee R^- = R$; O^I 为算子; 关系表 V_l 可以是临时表或基表; $R_{j+}^- (R_{j+}^+)$ 为基表 $R^- (R^+)$ 第 $j^- (j^+)$ 个属性, $S_k (S_{k'})$ 为势估算值. 由于双目算子至多操作一个临时表^[11], 因此 O^I 操作基表 R^- 和临时表 V_l (或基表 R^+) 的属性 R_{j-}^- 和 R_{j+}^+ .

行 3~8 将有序集 \mathcal{S} 划分为五元组聚类的集合 \mathcal{T} , DM4 使用标志位设置算子的执行策略. 行 5 利用该位及后序遍历的有序性确定 \mathcal{T} 中聚类的时态偏序关系.

DM4 将同类算子放在算子块内阻塞执行以提高效率. 利用算子块内并发执行, 行 9~25 用代价最大的块内算子近似估算算子块的总 I/O 代价. 其中函数 $\max(s_i \cdot S_k, s_i \cdot S_{k'})$ 用零值处理单目算子下的空参数, 双目算子的 I/O 代价用算子的两操作对象之中较小者 $\min(s_i \cdot S_k, s_i \cdot S_{k'})$ 进行比较^[10]. 变量 T 记录了 O^I 之前已访问的 R 属性并消除聚类间重复属性, $\bar{T} = R - T$ 被用于过滤临时表 V^J 在分区存储下多余的 R 属性.

如图 1 所示, 上例中基表 $L_i = \{L_1, L_2, \dots, L_{16}\}$ 被 DPOS 算法划分为 $\{L_9, L_1, \{L_6, L_7\}, \text{其它}\}$ 4 个分区.

4 聚类模型

4.1 代价分析

算法 DPOS 利用查询 Q 的左子树分解得到基表 R 的时态分区 C , 其中任意时态访问聚类 R^I 包含 R 中被 O^I 所在算子集 (下称算子集 O^I) 访问的属性集. 若 R^I 和 R^J 分属于 C 中两元素 c^I 和 c^J , 则 c^J 刻画了算子集 O^J 作用于基表 T^J 和关系表 V^J , 且分别各对应元组势 L^J 和 S^J . 若基于 C 分区存储 R , 则 O^J 的执行代价由子表集 $\Sigma_{I < J} R^I$ 、关系表 T^J 和 V^J 的 I/O 代价 C_{i0} 与聚类间元组寻址代价 $L(R^I, R^J)$ 构成. 下面依据 T^J 和 V^J 与 $\forall I < J, R^I$ 的关系, 分析基于 C 分离存储 R^I 和 R^J 时 O^J 的执行代价 T_C^J 和 NSM 下 O^J 的执行代价 T_R .

4.1.1 执行代价 T_R

A.1 O^J 为单目算子

若 $R \cap V^J \neq \emptyset$, O^J 操作临时表 V^J 的属性集 R^J ;

否则, V^j 为空且 O^j 操作基表 $T^j = R$ 的属性集 R^j .

$$T_R = \begin{cases} C_{10}(O^j, R \cup V^j, S^j), & R \cap V^j \neq \emptyset \quad (1a) \\ C_{10}(O^j, R, L^j), & R \cap V^j = \emptyset \quad (1b) \end{cases}$$

B.1 O^j 为双目算子

若 $R \cap V^j \neq \emptyset$, 则 O^j 操作临时表 V^j 的属性集 R^j 和基表 T^j ; 否则 O^j 操作基表 $T^j = R$ 的属性集 R^j 和 V^j .

$$T_R = \begin{cases} C_{10}(O^j, T^j, L^j, R \cup V^j, S^j), & R \cap V^j \neq \emptyset \quad (2a) \\ C_{10}(O^j, R, L^j, V^j, S^j), & R \cap V^j = \emptyset \quad (2b) \end{cases}$$

在此基础上, 可引入寻址代价 $L(R^i, R^j)$, 通过分析 R^i 与操作对象 T^j 或 V^j 的模式包含关系计算 T_C^{IJ} .

4.1.2 执行代价 T_C^{IJ}

A.2 O^j 为单目算子

A.2.1 $R \cap V^j = \emptyset$

关系表 V^j 为空且 T^j 为操作对象. 若 $R^i \subset T^j$, 则 O^j 通过 T^j 蕴含的 R^i 元组的标识操作 R^j 元组; 否则 O^j 依据 $R \cap T^j$ 操作 R^j 元组而与 R^i 无关. T_C^{IJ} 为

$$\begin{cases} C_{10}(O^j, T^j, L^j) + L^j L(R^i, R^j), & R^i \subset T^j \quad (3a) \\ C_{10}(O^j, T^j, L^j), & R^i \not\subset T^j \quad (3b) \end{cases}$$

A.2.2 $R \cap V^j \neq \emptyset$

O^j 依据 V^j 操作 R^j 元组. 类似于 A.2.1, T_C^{IJ} 为

$$\begin{cases} C_{10}(O^j, V^j, S^j) + S^j L(R^i, R^j), & R^i \subset V^j \quad (4a) \\ C_{10}(O^j, V^j, S^j), & R^i \not\subset V^j \quad (4b) \end{cases}$$

B.2 O^j 为双目算子

B.2.1 $R \cap V^j = \emptyset$

基表 T^j 蕴含 R 元组. 若 $R^i \subset T^j$, 则 O^j 用 T^j 蕴含的 R^i 元组对 R^j 元组寻址; 反之与 R^i 无关. 故 T_C^{IJ} 为

$$\begin{cases} C_{10}(O^j, T^j, L^j, V^j, S^j) + L^j L(R^i, R^j), & R^i \subset T^j \quad (5a) \\ C_{10}(O^j, T^j, L^j, V^j, S^j), & R^i \not\subset T^j \quad (5b) \end{cases}$$

B.2.2 $R \cap V^j \neq \emptyset$

关系表 V^j 蕴含 R 元组. 类似于 B.2.1, T_C^{IJ} 为

$$\begin{cases} C_{10}(O^j, T^j, L^j, V^j, S^j) + S^j L(R^i, R^j), & R^i \subset V^j \quad (6a) \\ C_{10}(O^j, T^j, L^j, V^j, S^j), & R^i \not\subset V^j \quad (6b) \end{cases}$$

如算法 DPOS 步 16 所述, 算子集 O^j 中各算子对关系 R 的属性采取相同处理, 这造成 O^j 将子表 R^i 的所有属性作为整体进行输出, 故式(3)~(6)中 V^j 和 T^j 与 R^i 只有包含和相离关系. 此外, 算法中变量 T 仅记录了 O^j 执行前被访问的子表属性, 因此上述 T_R 和 T_C^{IJ} 可依据彼此包含关系对操作对象做不同处理.

4.2 有效增益

前面将分区模式下算子的执行代价分成关系表

的 I/O 代价和子表间元组的寻址代价两部分. 单查询下的聚类是否分区存储, 主要取决于 NSM 与 CSM 下算子执行代价的差异. 由于覆盖 U 下两聚类 A^i, A^j 可转化为 $A^i, A^j - A^i$, 因此下面仅就分区 C 讨论代价差异.

定义 4. 给定基表 R 在查询 Q 下的时态分区 $C = \{R^0, R^1, \dots, R^K\}$, 则 $R^i, R^j \mid I < J$ 被分割为子表的有效增益为

$$D^{IJ} = T_R - T_C^{IJ} \quad (7)$$

依据 O^j 的类型、 R 与 V^j 的关系、 R^i 与 O^j 操作对象 V^j 或 T^j 的关系, T_C^{IJ} 被分为式(3a)~(6b)所示的 8 种情况, 且由于 T_C^{IJ} 蕴含 T_R 的分类, D^{IJ} 也对应地存在 8 组分析公式. 式(7)将分离 R^i 和 R^j 前后执行 O^j 的代价差定义为两聚类的有效增益. 据此可得下面引理.

定理 1. 给定基表 $R = \{A_1 \dots A_i A_{i+1} \dots A_j \dots A_n\}$, 其中属性集 $R^i = \{A_s \mid 1 \leq s \leq i\}$ 和 $R^j = \{A_i \mid 1 < i \leq j\}$ 为 R 在 C 中两聚类, 且 $R^\emptyset = \{A_s \mid j \leq s \leq n\}$, 则分离 R^i, R^j 的有效增益取决于算子集 $O^{\max(I, J)}$ 请求的元组分布和二者的寻址代价.

依据时态访问的有序性, 选取较晚执行的算子集 $O^{\max(I, J)}$, 调用式(7)计算 D^{IJ} 可得定理 1. 迭代计算任意两聚类间的有效增益可生成 R 在单查询 Q 下的最优分区, 下面将这一思想拓展到多个查询的情形.

4.3 聚类质量

若给定查询样本集 $Q = \{Q_1 \dots Q_m\}$, 为生成基表 R 的最优分区, 我们针对 Q 构造 R 的属性聚类质量作为 R 中属性聚类的判别标准. 对于 Q 中任意查询 Q_k , 应用 DPOS 算法可得到基表 $R = \{A_1 A_2 \dots A_n\}$ 的时态分区 $C_k = \{c_k^0, c_k^1 \dots c_k^{K_k}\}$ (以下相关符号均以下标区别 Q 中不同查询), 进而生成 R 的属性映射矩阵 $P = (R, Q)$:

$$P = (A_1, A_2, \dots, A_n) = \begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_m \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ P_{m1} & P_{m2} & \dots & P_{mn} \end{bmatrix} \quad (8)$$

其中 $A_i \mid 1 \leq i \leq n$ 为基表 R 定义中的第 i 个属性, $Q_k \mid 1 \leq k \leq m$ 为集合 Q 中第 k 个查询, $0 \leq P_{ki} \leq K_k$ 为 A_i 在 Q_k 下的聚类 $c_k^{P_{ki}}$ 的序号. 式(8)中矩阵行向量刻画了 Q_k 对 R 各属性的时态划分. 由定理 1 易得如下推论.

推论 1. 在 $\forall Q_k \in Q$ 下, 分离 $\forall A_i, A_j \in R$ 的有

效增益 $D_k(i, j)$ 与聚类序号 $I = P_{ki}, J = P_{kj}$ 有如下关系:

$$D_k(i, j) = \begin{cases} D_k^{\min(I, J), \max(I, J)}, & I \neq J \\ -S_k^I L(R_k^I, R_k^I), & I = J \wedge R \cap V_k^I \neq \emptyset \\ -L_k^I L(R_k^I, R_k^I), & I = J \wedge R \cap V_k^I = \emptyset \end{cases} \quad (9a)$$

$$D_k(i, j) = \begin{cases} -S_k^I L(R_k^I, R_k^I), & I = J \wedge R \cap V_k^I \neq \emptyset \\ -L_k^I L(R_k^I, R_k^I), & I = J \wedge R \cap V_k^I = \emptyset \end{cases} \quad (9b)$$

$$D_k(i, j) = \begin{cases} -S_k^I L(R_k^I, R_k^I), & I = J \wedge R \cap V_k^I \neq \emptyset \\ -L_k^I L(R_k^I, R_k^I), & I = J \wedge R \cap V_k^I = \emptyset \end{cases} \quad (9c)$$

其中 R_k^I 和 R_k^J 各自对应元素 c_k^I 和 c_k^J 中 R 的属性聚类, Q_k 将属性 A_i 和 A_j 分别划分到聚类 R_k^I 和 R_k^J : (1) $I \neq J$ 时, 由定理 1, R_k^I 和 R_k^J 分别包含 Q_k 中算子集 O_k^I 和 O_k^J 映射的属性, 故可得式 (9a). (2) $I = J$ 时, A_i 和 A_j 同时被 O_k^I 访问, 若 A_i 和 A_j 被分离, 则 O_k^I 访问的聚类 R_k^I 被分成两个相离的子表, 执行时二者被 O_k^I 以非阻塞方式交迭访问, 二者之间的寻址次数主要由 S_k^I 或 L_k^I 决定, 式 (9b) ~ (9c) 以 R_k^I 的分布估算该寻址代价.

定义 5. $\forall A_i, A_j \in R$ 在 Q 下时态聚类距离为

$$D(i, j) = \exp\left(\frac{1}{m} \sum_{k=1}^m D_k(i, j)\right) \quad (10)$$

基于式 (8) 和式 (9) 对任意两属性 A_i, A_j 有效增益的度量, 利用 $\forall Q_k \in Q$ 下 $D_k(i, j)$ 的算术均值, 式 (10) 指数化了查询集 Q 下分离任意两属性的性能增益.

5 模型求解

5.1 明细代价

综上所述, 依据时态访问聚类与操作对象的逻辑模式的关系, 属性间的时态聚类距离被表示为不同存储模式下算子 I/O 代价和子表间元组寻址代价的多项式. 文献[13]基于块式磁盘对几种连接算法建立明细 I/O 代价模型, 综合考虑了块式磁盘的

三大要素: 寻道延迟、旋等延迟和传输延迟, 并将算子的 I/O 代价表示为 $N_S T_S + N_R T_L + N_X T_X$. 利用算子操作对象的磁盘块数及其分布估计, 我们基于 SCSI 磁盘将算子的 I/O 代价 C_{10} 和寻址代价 $L(R^I, R^J)$ 表示为上述三大参数的多项式. 表 1 为明细代价的相关参数及其默认值.

5.1.1 单目算子下的 C_{10}

在式 (1) 和式 (3)、(4) 的 C_{10} 中, 算子集 O^J 仅操作临时表 $R \cup V^J$ 、基表 R, T^J 和关系表 V^J 其中之一, 系统通过统计可以估算给定数目的元组所占用的磁盘块数 (如 L^J 条 T^J 元组占用 $P(T^J, L^J)$ 块). 下面以 R 和 L 指代上述操作对象及其元组势分析 C_{10} , 假设内存共有 $M+O$ 页, 其中 $O=1$ 页作为输出缓冲.

A.1 简单扫描

当算子集 O^J 由选择、计数或求和等物理操作符组成时, 关系表 R 中有 $P(R, L)$ 个磁盘块被请求, 则 O^J 仅需扫描一次 R . 由于可按柱面组织 R 的 $P(R, L)$ 个磁盘块, 且有 M 页可操作内存, 故三大系数为

$$N_S = \lceil P(R, L) / M \rceil,$$

$$N_R = M \lceil P(R, L) / M \rceil / c,$$

$$N_X = P(R, L).$$

B.1 M 路排序

考虑 M 路归并排序 R , 给定 M 页可操作内存, 算法需要执行 $\lceil \ln_M P(R, L) \rceil$ 轮; 每轮排序 $\lceil P(R, L) / M \rceil$ 次, 首次访问连续柱面上的 M 块; 后续每轮各访问间隔柱面的 M 块. 考虑到柱面间隔和磁道切换, 故

$$N_S = \lceil P(R, L) / M \rceil +$$

$$M \lceil P(R, L) / M \rceil (\lceil \ln_M P(R, L) \rceil),$$

$$N_R = M \lceil P(R, L) / M \rceil / c +$$

$$M \lceil P(R, L) / M \rceil (\lceil \ln_M P(R, L) \rceil),$$

$$N_X = \lceil \ln_M P(R, L) \rceil \times P(R, L).$$

5.1.2 双目算子下的 C_{10}

在式 (2) 和式 (5)、(6) 的 C_{10} 中, 算子集 O^J 操作临时表 $R \cup V^J$ 、基表 R, T^J 和关系表 V^J 中任意两个. 下面以 R 和 V 分别指代 O^J 的两个操作对象分析 C_{10} , 其元组势分别为 L 和 S , 不妨设 M 页内存且 $P(R, L) \leq P(V, S)$.

A.2 嵌套连接

以效率较高的块连接^[10]为例, 它将 M 页内存分为两部分, M_R 页缓存外关系 R, M_V 页缓存 V , 每次读取 M_R 个连续的 R 块与 V 连接, 执行 $\lceil P(R, L) / M_R \rceil$ 次后完成连接. 依照柱面组织 R 和 V , 则三大系数为 $N_S = \lceil P(R, L) / M_R \rceil + P(V, S) \lceil P(R, L) / M_R \rceil / c,$

表 1 明细代价参数

符号	参数意义	默认值
b	系统中磁盘块及内存页的大小	8KB
c	柱面平均磁盘块数 (以 b 计算)	≈ 120
T_S	磁盘平均寻道时间	8.9ms
T_L	磁盘平均旋转等待时间	4.2ms
T_X	磁盘平均单块传输时间	0.8ms
$M+O$	系统内存的总页数 (O 为输出缓冲)	variable
M_T	分配给关系表 T 的内存页数	variable
$\ R\ , R $	关系表 R 的元组势和属性总字节数	variable
T^I, V^I	算子集 O^I 操作基表和中间表	variable
L^I, S^I	算子集 O^I 操作的关系表元组势	variable
$G(T^I, A)$	关系表 T^I 在属性 A 上的取值个数	variable
$P(T^I, L^I)$	关系表 T^I 中 L^I 条元组所占磁盘块数	variable
$L(R^I, R^J)$	由子表 R^I 到 R^J 的元组寻址代价	variable
N_S	算子 IO 请求的寻道次数	variable
N_R	算子 IO 请求的旋转等待次数	variable
N_X	算子 IO 请求的磁盘块数	variable

$$N_R = M_R \lceil P(R, L) / M_R \rceil / c + P(V, S) \lceil P(R, L) / M_R \rceil / c,$$

$$N_X = P(R, L) + P(V, S) \lceil P(R, L) / M_R \rceil.$$

B.2 归并连接

该算子首先排序两对象, 然后分块连接和归并输出. 相应的三大系数分别为排序与连接的系数和, 其中归并排序已经在 B.1 中分析. 若 R 和 V 在连接属性 A 上有序且取值服从均匀分布, 则 $P(V, S) / G(V, A)$ 和 $P(R, L) / G(R, A) + 1$ 分别为内外连接区的粒度. 一般有 $M < \min(P(R, L) / G(R, A) + 1, P(V, S) / G(V, A))$, 且其中 $P(R, L) / G(R, A) < P(V, S) / G(V, A)$. 假定为 R 分配 $M_R = \lceil P(R, L) / G(R, A) \rceil$ 页内存、为 V 分配 $M_V = M - M_R$ 页内存, 则连接被分段完成: 每段中一个 R 区(在 A 上等值的 M_R 个内存页)依次与 M_V 个 V 的内存页连接; 若到达 V 的尾页则读入 M_V 个 V 的新页, 若到达 R 的区尾则交迭读入新区直至完成连接. 该连接的三大系数为

$$N_S = \lceil P(R, L) / M_R \rceil + \lceil P(V, S) / M_V \rceil,$$

$$N_R = M_R \lceil P(R, L) / M_R \rceil / c + M_S \lceil P(V, S) / M_S \rceil / c,$$

$$N_X = P(R, L) + P(V, S).$$

C.2 散列连接

两个对象均不能单独容于内存时, GRACE 散形较其它散列方法具有更稳定的连接效率^[13]. 若 R 和 V 在 A 上服从均匀分布, 算法由两阶段组成:

前一阶段依据散列函数将各对象划分到 $NB = \lceil P(R, L) / M - I_P \rceil$ 个桶中, 其中 I_P 为输入的缓冲页数, 每个桶至多占用 $B = \lceil (M - I_P) / NB \rceil$ 个磁盘块; 后一阶段逐次读入 R 桶对应地与 S 桶连接, 并以 $I_M = M - B$ 页内存缓冲 V . 上述过程所要求的 $M > \sqrt{P(R, L) + I_P}$ 一般可被满足, 故

$$N_S = 2 + 2NB,$$

$$N_X = 2P(R, L) + 2P(V, S),$$

$$N_R = NB \lceil B/c \rceil + I_P (\lceil P(R, L) / I_P \rceil + \lceil P(V, S) / I_P \rceil) / c + I_M \lceil P(V, S) / I_M \rceil / c.$$

上面分析了几类常见算子的 I/O 代价, 基于索引的算子以索引深度为单位随机访问磁盘块, 其代价可由 A.1、B.2 和下节的 B.3 叠加得到^[13].

5.1.3 元组寻址代价

在式(3)~(6)中我们将分析对象 R 分成 K 个属性相离的子表讨论算子的执行代价. 为保证不同子表间元组的逻辑对应关系, 我们使用 DM4 的文件组存储子表. 各子表由 R 中的元组经过拆分得到, 各子表的元组由一个递增的 64 位 ID 域标识: 在定长属性的子表中, ID 用于计算元组所在的磁盘块号和块内偏移; 变长属性的子表利用稀疏 B 树索引各块中起始 ID, 磁盘块首部记录各 ID 对应元组的块内偏移, 如图 2 所示.

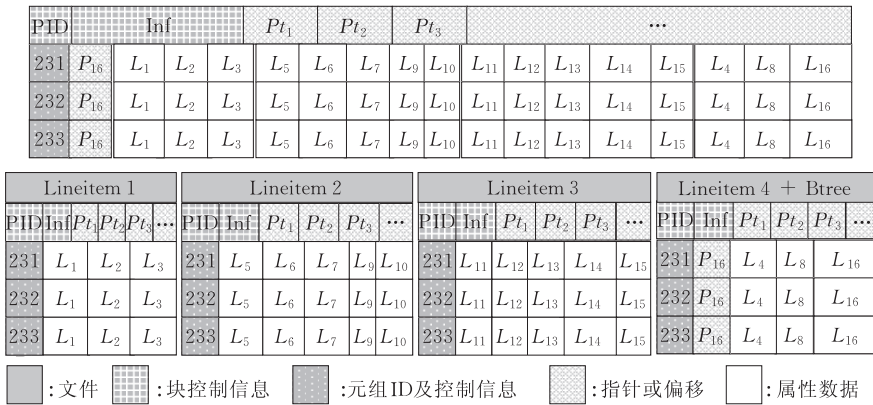


图 2 TPC-H 基准中基表 Lineitem 的存储示意图(基表被存储为 4 个文件 Lineitem1~4, 前三者存储定长子表, Lineitem4 为变长子表, 各子表之间的元组通过 PID 关联)

图 2 中子表 Lineitem1~4 存储在顺序文件中, 一共包含基表的 16 个属性, 元组 ID 确定了各子表间元组的逻辑对应关系. 依据子表 R 是否包含变长属性, 式(3)~(6)中 R 对 V 的寻址代价 $SL(V, R)$ 可分为两种情形.

A.3 定长子表

若 R 中不包含变长属性, 则子表 V 中的 S 个元

组在 R 中的磁盘块号和块内偏移分别为 $Rid \div \lfloor b/|R| \rfloor$ 和 $Rid \bmod \lfloor b/|R| \rfloor$, 其中 Rid 为每个元组的 ID. 与 C_{10} 不同, 这里访问的 R 元组与关系表 V^j 或 T^j 中蕴含的 V 元组一一对应. 若 S 个元组在 V 中连续存储, 则 $SL(V, R)$ 为 $\lceil S|R|/b \rceil$ 块 R 磁盘块的简单扫描代价; 否则 $P(R, S)$ 个 R 的磁盘块被随机访问, 且其寻址代价为 $P(R, S)(T_S + T_L + T_X)$.

B.3 变长子表

若 R 中包含变长属性,则以稀疏 B 树索引 R 的磁盘块.若变长子表 R 中单个元组平均占用 $|\hat{R}|$ 字节, B 树中指针占用 8 字节,填充因子为 f ,则树深不超过 $MD = \lceil \ln_{\lceil f(b/16-1) \rceil} (\|R\| |\hat{R}| / b) \rceil$. 我们将树根常驻内存,对子表的一块进行随机访问需要首先读入 $MD-1$ 个树结点磁盘块,则其 I/O 代价为 $t_R = (T_s + T_L + T_X)MD$. 若 S 个元组在 V 中连续存储,则总寻址代价为一块随机访问与 $\lceil S|\hat{R}|/b \rceil - 1$ 块简单扫描的 I/O 代价和;否则为 $P(R, S)$ 块随机访问的 I/O 代价 $P(R, S)t_R$. 在 DM4 中,子表 Lineitem4 的一个元组平均占用 68 字节,若 f 取 0.8 则 3 层上述 B 树可以索引 8.18×10^9 个元组,这相当于 3TB 的 TPC-H 元数据量.

5.2 初始等价关系

利用明细代价量化属性的时态聚类距离 $D(i, j)$, 则对于给定查询集 Q ,基表 R 的属性映射矩阵 P 和时态距离 $D(i, j)$ 构成时态访问模型 $PD = \langle P, D \rangle$: 行向量 $Q_k = (P_{k1}, P_{k2}, \dots, P_{kn}) | 1 \leq k \leq m$ 刻画 R 各属性在 $\forall Q_k \in Q$ 下的时态映射值、列向量 $A_i = (P_{1i}, P_{2i}, \dots, P_{mi})^T | 1 \leq i \leq n$ 为属性 $A_i \in R$ 在各查询中的时态映射值;任意两列向量 A_i 和 A_j 之间的距离由 Q 下属性的时态聚类距离 $D(i, j)$ 度量. 模型求解以列向量空间 $P(R) = \{A_i | 1 \leq i \leq n\}$ 为对象空间,以 $D(i, j)$ 为聚类距离,生成 R 的一个硬划分 $R(Q) = \{C^0, C^1, \dots, C^K\}$,使 Q 的总执行代价最小.

PD 的求解是个 NP 问题,可通过无监督聚类分析解决^[14]. 但已有的基于目标函数的方法主要面向数值型对象,依赖聚类中心或质点度量对象与聚类的距离,且聚类质量在很大程度上取决于聚类原型^[15]. 由于对象 A_i 由非数值型序号构成,因此 PD 不满足完备性. 近年来,由于适合解决不精确、不确定和不完备的问题,粗集理论^[16-17] 被引入到聚类领域^[15,18],本文基于粗糙等价关系^[15]、利用对象邻域实现 PD 的聚类.

对于 PD 的对象空间 $P(R)$ 中任意对象 A_i, A_j , 距离 $d(A_i, A_j) = D(i, j)$ 独立于其它对象 $\forall A_l \neq A_i, A_j$, 故每个 A_i 可以确定一个等价关系 E_i :

$$P(R)/E_i = \{P_i, \overline{P_i}\} \quad (11)$$

其中 P_i 为 A_i 的所有邻域对象之集, $\overline{P_i} = P(R) \setminus P_i$ 为 P_i 的补集, P_i 由 $P(R)$ 中对象与 A_i 的相对距离确定. 而由式(10)和定理 1 可知:若 $d(A_i, A_j) > 1$, 则将 A_i, A_j 分布于不同子表能减少 Q 中算子的 IO 代价. 因此对于任意 A_i , 我们用邻域阈值 $Td_i = 1$ 确定 P_i ,

即

$$P_i = \{A_j | \forall A_j \in P(R), d(A_i, A_j) \leq 1\} \quad (12)$$

上述 $P(R)$ 中 n 个对象 A_i 各确定一个等价关系 E_i , 并将 $P(R)$ 划分为两个等价类 $P(R)/E_i = \{[A_i]_{E_i}, \overline{[A_i]_{E_i}}\}$, 我们称等价关系集 $\mathcal{E} = \{E_i | 1 \leq i \leq n\}$ 为一个等价关系族, 称 $\mathcal{W} = (P(R), \mathcal{E})$ 为 $P(R)$ 在 \mathcal{E} 下的一个关系系统.

5.3 迭代求精

若基于不可辨识关系 $ind(\mathcal{E}) | [A_i]_{ind(\mathcal{E})} = \bigcap_{E \in \mathcal{E}} [A_i]_E$ 分类 $P(R)$ 中对象, 则 $\forall A_i, A_j \in P(R)$ 属于同一类块当且仅当在 $\forall E_k \in \mathcal{E}$ 下 A_i 和 A_j 均属于同一等价类. 这种过于严格的划分造成类块的粒度过小, 而不能反映对象之间的相似性. 为此, 我们定义不可辨识度放松对象间的相似度要求, 通过阈值 Th 迭代逼近一个稳定分类.

定义 6. 在关系系统 \mathcal{W} 中, $\forall A_i, A_j \in P(R)$ 之间的辨识因子和不可辨识因子分别为

$$\delta_k(A_i, A_j) = \begin{cases} 1, & (x_i \in [A_k]_{E_k} \wedge x_j \notin [A_k]_{E_k}) \vee \\ & (x_i \notin [A_k]_{E_k} \wedge x_j \in [A_k]_{E_k}) \\ 0, & \text{其它} \end{cases} \quad (13a)$$

和

$$\overline{\delta}_k(A_i, A_j) = \begin{cases} 1, & (x_i \in [A_k]_{E_k} \wedge x_j \in [A_k]_{E_k}) \\ 0, & \text{其它} \end{cases} \quad (14a)$$

进而, A_i 和 A_j 之间的不可辨识度为

$$\gamma(A_i, A_j) = \frac{\sum_{k=1}^{|P(R)|} \overline{\delta}_k(A_i, A_j)}{\sum_{k=1}^{|P(R)|} \overline{\delta}_k(A_i, A_j) + \sum_{k=1}^{|P(R)|} \delta_k(A_i, A_j)} \quad (15)$$

在式(13)中, 若等价关系 E_k 将 A_i 和 A_j 入不同的等价类, 则两者在 E_k 下是可辨识的, 并将其辨识因子置 1; 否则, 两者是非可辨识的并置 0. 在式(14)中, 若 A_i 和 A_j 均在 A_k 的邻域中, 则两者在 E_k 下是不可辨识的, 不可辨识因子置 1; 否则置 0. 特别地, 若 $A_i \notin [A_k]_{E_k} \wedge A_j \notin [A_k]_{E_k}$, 则 $A_i \in \overline{[A_k]_{E_k}} \wedge A_j \in \overline{[A_k]_{E_k}}$, 此时 A_i 和 A_j 均在 A_k 的邻域外, 两者的辨识性不能确定, 故不能用于判定两者的相似性, 如式(15)分子所示.

通过 $P(R)$ 中两两对象在 \mathcal{E} 下的不可辨识度, 可得到一个初始不可辨识度方阵 $\mathbf{I}^{(0)} = \{\gamma(A_i, A_j)\}_{n \times n}$. 利用不可辨识阈值 Th , 可以生成 \mathcal{E} 的 1 阶求精等价关系族 $\mathcal{E}^{(1)} = \{E_i^{(1)} | 1 \leq i \leq n\}$, $\mathcal{E}^{(1)}$ 中每个等价关系 $E_i^{(1)}$ 依据 A_i 的 1 阶邻域 $P_i^{(1)}$ 二分 $P(R)$:

$$P(R)/E_i^{(1)} = \{P_i^{(1)}, \overline{P_i^{(1)}}\} \quad (16)$$

后基于分类中块间和块内的对象时态聚类距离计算分类索引. 计算方阵和索引的时间复杂度均为 $O(n(n+1)/2)$, 迭代次数理论极值为 $\lceil nT_h \rceil$, 但实践中一般有 $r \leq 6^{[15]}$, 故模块复杂度为 $O(rmn(n+1))$.

7 实验结果

实验分别针对分类性能、仿真和真实数据的执行效率进行测试. 测试硬件平台为曙光服务器, 含 $2 \times$ Xeon 2.4GHz CPU、4GB 内存和 8×36.9 GB SCSI 磁盘, 参数见表 1; 软件环境为 Win2003 和 DM4 数据库.

7.1 分类算法的性能

DSS 应用中关系表的维数一般不大于 256, 我们选取 Neyman-Scott 方法^[15]生成仿真数据, 比较上述 IIADDI (Iterative indiscernibility with ADDI) 分类和分别基于平均联结 (AL-AHC) 和完全联结 (CL-AHC) 的两种层次分类^[19]以及相对近似分类 (RAC)^[15]. 实验用 Neyman-Scott 方法生成 256 个二维对象及其基于欧氏距离的距离方阵; 然后按一定比率随机修改方阵数据以破坏距离的三角不等性. 我们将执行结果相对于真实值的对象正确分类比作为分类精度进行比较, 且 RAC 取阈值为 $1/6, \dots, 1/2$ 时分类的平均精度.

图 4、图 5 反映了种子点分别为 3 和 6 时分类精度相对于修改率的变化情况. 图中显示了若满足三角不等式距离的对象比率较小, 如 $Ratio > 0.1$ 时, IIADDI 和 RAC 的分类效率较另两种方法具有显著的提高; 而且 IIADDI 的 ADDI 索引有效解决了 RAC 分类的参数选择问题. 此外, 图 5 中精度拐点表明了修改比率比类块密度 (种子数) 对算法精度的影响更为显著.

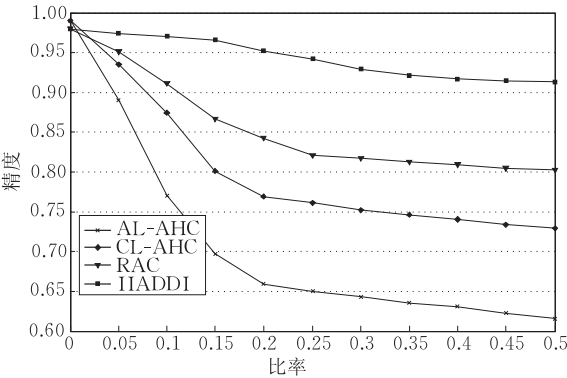


图 4 种子点为 3 时的分类精度

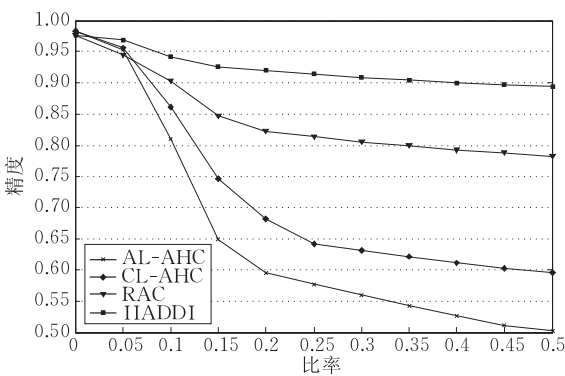


图 5 种子点为 6 时的分类精度

7.2 TPCB 仿真验证

上述实验说明 IIADDI 算法解决非三角不等距离的分类问题具有较高效率. 基于此, 本节基于 TPCB 基准, 测试我们的 TIP (Temporal access model with IIADDI Partitioning) 分区系统和 CSP^[6]、VPC^[5] 以及 Autopart^[2] 的查询执行和优化时间. 实验分为三个阶段: 首先利用 DBGEN 工具生成数据规模分别为 1, 10, 30 和 100 的 TPCB 元数据并载入数据库; 然后利用 QGEN 工具以默认参数生成 22 个查询, 并用 4 种系统分别生成垂直分区方案; 最后依据各种方案整理关系表并执行三套重复查询以比较查询效率. 其中, 以三套查询的平均执行时间衡量各查询. 图 6、图 7 分别列举了数据规模为 10 和 100 时, 22 个查询中的 10 个查询在上述 4 种分区结果和 NSM 下的运行时间, 前者不用索引比较元数据的优化效率、后者比较总效率.

上述待测分区系统均基于文件组, 以 ID 计算元组偏移实现分区之间元组的逻辑对应. 其中, TIP 的参数 c_{max} 设为磁盘数量 8, 依据图 3 分类模块的结果, 第三阶段依次执行如下步骤: 创建文件组并在其内建立对应于各类块的文件, 使得文件分布于不同磁盘; 按照 5.1.3 节的 ID 构造法, 依据分类结果分别在文件上创建子表; 分割分析表并将元组存入子表. Autopart 采用优化器迭代估算法完成分区. 如图 6 所示: 由于 CSP 和 VPC 直接以元组数量估计扫描和连接的代价, 故不能反映具备迭代特性的算子 (如 Q1 的分组、排序以及 Q18 和 Q21 的连接算子), 导致这些算子中的属性被分离存储, 从而造成庞大的寻址数量; Autopart 基于总体代价分析, 使得其分区结果能够保证最大代价的查询得到尽可能的最优化, 如 Q18 使得所有查询涉及的属性放入一个分区, 却因此增加了中等查询对无用数据的访问

(如 Q5、Q10),且涉及算子较多使得估算精确度受到限制.由于上述 4 种系统均在每个分区中创建 ID,同规模的系统之间空间差异不足 3.8%,因而试验没有讨论空间利用率.图 7 说明上述不足在有索引、大数据量时更明显.

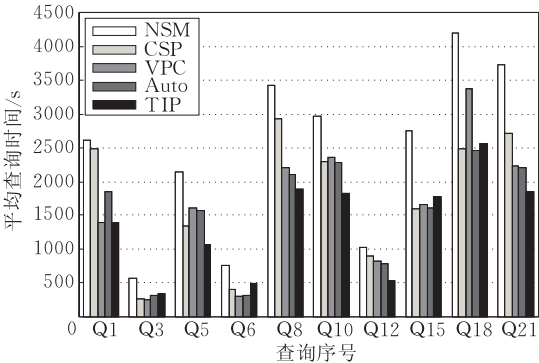


图 6 数据规模为 10 的无索引查询时间

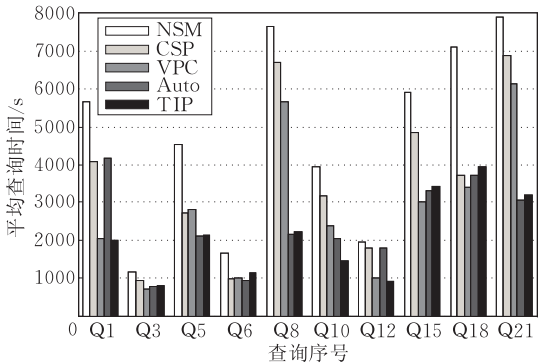


图 7 数据规模为 100 的有索引查询时间

表 2 TPC-H 各阶段执行时间及查询总加速比

系统	规模/G	载入时间/s	分区时间/s	整理时间/s	加速比/%
CSP	10	2023	277	821	21.3
	30	6045	276	2365	18.4
VPC	10	2023	382	817	32.9
	30	6045	375	2352	39.1
Auto	10	2023	1121	826	35.7
	30	6045	1056	2371	43.0
TIP	10	2023	291	819	45.7
	30	6045	268	2368	58.3

表 2 概括了 4 种系统的数据载入、分区计算、数据整理时间及相对于 NSM 的总体加速比.如表中所示:由于 CSP 使用分区树缩减分区组合空间,故比 VPC 快,TIP 的矩阵迭代介于二者,Autopart 的优化器迭代最慢;载入和整理阶段占用的时间随规模呈线性增长,而 4 种系统生成分区方案所占用的时间保持恒定;4 种系统的数据载入时间相同,数据规模较大(大于 30)时载入阶段的时间占用率超过 60%、各系统整理时间的差异不足 2%.此外,加速

比一栏所示的结果与图 6、图 7 的测试结果相吻合.

7.3 真实数据集测试

本节测试基于某省出口退税预警系统,对 VPC、Autopart、TIP 三种分区方案下的查询执行时间进行比较.作为一个典型的数据仓库应用,该系统首先对 Ctais、防伪税控系统和出口退税审核系统三个数据源执行联机查询,然后基于汇总结果计算预警指标,最后通过预先设定的峰值判定企业出口退税申报的合法性.实验抽取 40 个查询作为样本空间,并将 Ctais 中规模最大的表 ZSXX(征收信息表,含 66 个属性、10 倍于其它表的元组势)作为分区对象,从两个侧面比较上述三类分区和 NSM 下的查询执行时间.

图 8 为累计查询时间随查询数量变化的情况,其中分区基于全体查询样本空间进行,执行时间以 4 为查询累加粒度.如图 8 所示,相较于 VPC 和 Autopart 的贪婪寻优,TIP 基于聚类思想迭代寻求全局最优分区,不仅查询累计耗时较少,且各查询耗时较为平均.

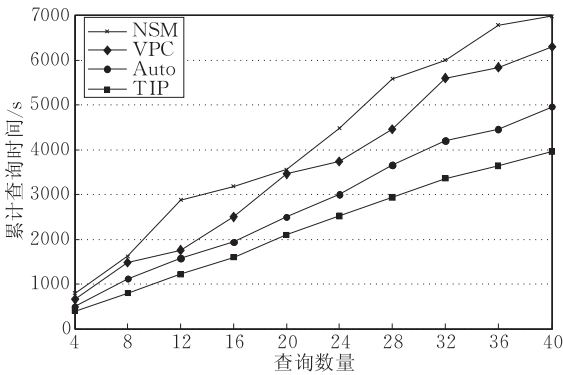


图 8 ZSXX 为 10G 时查询的累计执行时间

图 9 显示了数据规模递增过程中查询的平均响应时间.该测试挑选了查询样本空间中区别较为显著的 24 个查询作为测试对象,对不同规模下的数据

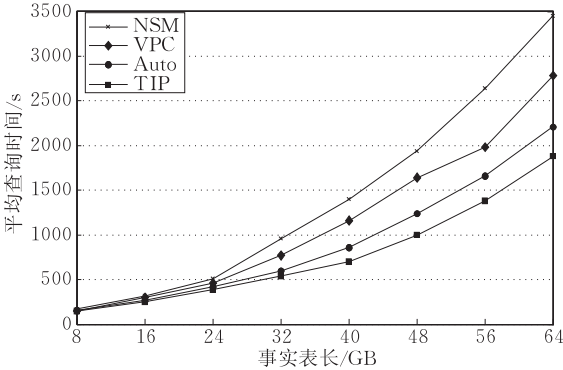


图 9 不同数据规模下 24 个查询的平均执行时间

执行分区算法,并将各分区下查询的平均响应时间作为比较指标.图中曲线的变化趋势较为有效地验证了 TIP 分区在处理大数据量查询时的优势.

8 总 结

为提高数据仓库的数据访问有效性,本文提出一种基于算子时态划分和粗集迭代求解的关系垂直分区方法.利用阻塞算子分解查询计划,用访问属性的算子集的平均代价度量其相似性,有效揭示了分析对象的属性被查询样本访问的密切度.以平均间距-直径索引评测和选取阈值参数,有效提高了分类质量.在付出较少优化时间的前提下,该方法能够显著减少样本集中各类查询的执行时间.实验证明:在决策支持应用中,该方法的效率优于同类的其它分区方法.

研究基于单一事实表假设,但现实中数据仓库往往存在多个较大的关系共存的情形,如实验中 Ctals 的申报信息 SBXX 与 ZSXX 规模相当.如何对多个事实表进行联合分区将作为我们下一步的研究方向.

参 考 文 献

- [1] Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. *ACM Sigmod Record*, 1997, 26(1): 65-74
- [2] Papadomanolakis S, Ailamaki A. Autopart: Automation schema design for large scientific databases using data partitioning//*Proceedings of the SSDBM*. Santorini Island, Greece, 2004: 383-392
- [3] Ng V T Y, Gorla N, Law D M, Kong C C. Applying genetic algorithms in database partitioning//*Proceedings of the SAC*. Melbourne EL, USA, 2003: 544-549
- [4] Ramamurthy R, DeWitt D J. A case for fractured mirrors//*Proceedings of the VLDB*. Hong Kong, China, 2002: 430-441
- [5] Agrawal S, Narasayya V R, Yang B. Integrating vertical and horizontal partitioning into automated physical database design//*Proceedings of the SIGMOD Conference*. Paris, France, 2004: 359-370
- [6] Huang Y F, Van C H. Vertical partitioning in database design. *Information Science*, 1995, 86(1-3): 19-35
- [7] Chu W W, Jeong I T. A transaction-based approach to vertical partitioning for relational database systems. *IEEE Transactions on Software Engineering*, 1993, 19(8): 804-812
- [8] Ailamaki A, DeWitt D J, Hill M D, Skounakis M. Weaving relations for cache performance//*Proceedings of the VLDB*. Rome, Italy, 2001: 169-180
- [9] Shao M, Schindler J, Schlosser S W. Clotho: Decoupling memory page layout from storage organization//*Proceedings of the VLDB*. Toronto, Canada, 2004: 696-707
- [10] Silberschatz A, Sudarshan, Forth H. *Database System Concepts*. 3rd Edition. New York, San Francisco, Washington, DC: McGraw Hill, 2000
- [11] Chaudhuri S. A overview of query optimization in relational systems//*Proceedings of the PODS*. Seattle, Washington, USA, 1998: 34-43
- [12] Agrawal S, Chaudhuri S, Das A, Narasayya V. Automating layout of relational databases//*Proceedings of the 19th International Conference on Data Engineering (ICDE)*. Bangalore, India, 2003: 607-618
- [13] Hass L M, Carey M J, Livny M, Shukla A. Seeking the truth about ad hoc join cost. *The Very Large Data Base Journal*, 1997, 6(3): 241-256
- [14] Furtado P. Experiment evidence on partitioning in parallel data warehouses//*Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP*. Washington, DC, USA, 2004: 23-30
- [15] Shoji H, Shusaku T. An indiscernibility-based clustering method with iterative refinement of equivalence relations//*Proceedings of the PKDD*. Cavtat-Dubrovnik, Croatia, 2003: 192-203
- [16] Pawlak Z. *Rough Sets: Theoretical Aspects of Reasoning About Data*. Dordrecht: Kluwer Academic Publisher, 1991
- [17] Ziarko W. Variable precision rough set model. *Journal of Computer and System Science*, 1993, 46(1): 39-59
- [18] Mitra S, Banka H, Pedrycz W. Rough fuzzy collaborative clustering. *IEEE Transactions on Systems, Man, and Cybernetics, B: Cybernetics*, 2006, 36(4): 795-805
- [19] Bezdek J C, Pal N R. Some new indexes for cluster validity. *IEEE Transaction on Systems, Man, and Cybernetics-Part B: Cybernetics*, 1998, 28(3): 301-315
- [20] Natsev A, Fuh G Y C, Chen W et al. Aggregate predicate support in DBMS//*Proceedings of the Australasian Database Conference*. Melbourne, Victoria, Australia, 2002: 111-120
- [21] Zhang Wen-Xiu, Wu Wei-Zhi et al. *The Theory of Rough Set and Its Method*. Beijing: Science Press, 2001 (in Chinese)

(张文修,吴伟志等.粗糙集理论与方法.北京:科学出版社,2001)



LI Wen-Hai, born in 1979, Ph. D. , lecturer. His current research interests include rough set theory, clustering analysis, decision support system, theory and implement of relation database management system.

FENG Yu-Cai, born in 1945, professor, Ph. D. supervisor. His current research interests include the theory and application of relation database and multi-media technology,

Background

This work is a part of the "Research on Automated Storage Design of Relation Database System", which is mainly supported by the National High Technology Research and Development Program (863 Program) under grant No.2004AA4Z3020. The project mainly focuses on the design and development of high-performance relational database management system. The research group including the authors of this paper has proposed several methods concern to some extent areas such as index selection, relational storage and view update based on rough set.

With the increasing requirements of mass data storage in

data mining and image processing.

MA Xiao-Ming, born in 1980, Ph. D. candidate. Her current research interests include the analytical and decision-making issues of the regional economy.

FU Quan, born in 1977, Ph. D. candidate. His current research interests include implement and application of security database management system.

HU Wen-Bin, born in 1977, Ph. D. , associate professor. His current research interests focus on decision support system and computer simulation.

DSS applications, high performance storage subsystems are expected to be adaptive with the query templates. In pursuit of this, various strategies and algorithms for efficiently storing and retrieving records have been developed to enhance the performance of the underlying data warehouse system. Rough set has attracted much research interests in the domain of clustering analysis. As for the DSS implement, the authors take relation database as the storage and query bases, and propose an indiscernibility based clustering method to automated partition the fact tables in terms of the query template set.