ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.361−371
DOI: 10.1360/jos180361

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

面向聚集查询的语义缓存技术*

蔡建宇+, 吴泉源, 贾 焰, 邹 鹏

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Semantic Cache Technology for Aggregate Queries

CAI Jian-Yu+, WU Quan-Yuan, JIA Yan, ZOU Peng

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4212030, E-mail: jianyucai@163.com, http://www.nudt.edu.cn

Cai JY, Wu QY, Jia Y, Zou P. Semantic cache technology for aggregate queries. *Journal of Software*, 2007,18(2):361−371. http://www.jos.org.cn/1000-9825/18/361.htm

Abstract: To process aggregate queries in massive database application efficiently, semantic cache technology is extended in this paper, which is mostly used in small scale database applications at present. Firstly, a formal semantic cache model for aggregate queries is proposed. Based on this model, a semantic cache system called StarCache is built. The key technologies of StarCache about aggregate query processing, cache replacement and consistency maintenance etc. are also discussed in this paper. StarCache has been integrated in StarTP, which is a parallel database middleware developed by a team of the National University of Defense Technology, and has been applied to a large national project.
Key words: aggregate query; semantic cache; query processing; cache replacement; consistency maintenance

摘 要: 将目前主要用于小规模数据库查询的语义缓存技术拓展到海量数据库的聚集查询中,以面向聚集查询的语义缓存形式模型为基础,构造了语义缓存 StarCache.详细讨论了 StarCache 中的聚集查询处理、语义缓存替换管理和一致性维护等技术.StarCache 已经集成在自主研发的并行数据库中间件 StarTP 中,并在一项大型国家工程中得到实际应用.
关键词: 聚集查询;语义缓存;查询处理;缓存替换;一致性维护
中图法分类号: TP311 文献标识码: A

随着海量数据库在关键业务中的应用不断增多,查询性能问题日益突出.海量数据库应用中存在大量聚集查询,对海量数据的聚集查询处理往往非常耗时,所以,提高聚集查询的处理效率是优化海量数据库系统查询性能的关键,已成为当前该领域的研究热点.现有的支持海量数据库聚集查询的方法主要有临时表、物化视图和 OLAP(on-line analytical processing)分析工具 3 类.临时表方法将常用的聚集查询预先执行结果存储在临时表中,当查询到来时,直接从临时表中提取结果.该方法对用户不透明,管理维护成本较高.物化视图方法通过在数

* Supported by the National Natural Science Foundation of China under Grant No.90412011 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2003AA111020, 2004AA112020, 2003AA115210, 2003AA115410 (国家高技术研究发展计划(863))

Received 2005-11-01; Accepted 2006-01-19

据库内部的视图中保存查询预先执行的结果来处理聚集查询,缩短查询响应时间.在物化视图研究领域已有很多工作,如文献[1−3]分别在 DB2,Oracle 和 SQL Server 数据库中利用物化视图来处理查询.物化视图方法存在的主要问题是:需要预先定义视图,不能根据应用需求随时作出动态调整;OLAP 分析工具往往需要重新组织数据才能充分发挥作用[4].因此,现有的 3 种方法都很难满足日益增多的海量应用的现实需求.

语义缓存机制[5−7]通过重用缓存的查询结果和相关查询描述优化查询,是提高查询数据库效率的有效手段.与其他技术相比,语义缓存具有一定的优势:对用户透明,不需要更改已有的业务;具有一定的自管理能力,可以减少人工管理成本.最初,语义缓存以客户/服务器系统中客户缓存的形式提出,缓存的内容一般都是“选择-投影-连接(select-project-join,简称 SPJ)”查询的结果[5−7].目前,语义缓存的主要应用领域包括 Web 应用、移动应用和 P2P.文献[8,9]通过缓存并重用 SPJ 查询结果来提高 Web 应用的性能.为了提高 Web 环境下 XML 查询处理的效率,人们将语义缓存机制引入 XML 查询,如 ACE-XQ[10];文献[11−14]对移动环境下的语义缓存技术进行研究,考察了移动应用中各种条件下的查询处理问题,基于语义缓存对移动客户端的 SPJ 查询进行处理;文献[15,16]利用节点(peer)请求之间的语义局部性来改善 P2P 系统的性能,支持通过 P2P 网络中多个节点的缓存构造查询结果,实现了分布的语义缓存系统.现有的语义缓存研究基本上是以 SPJ 查询为考察对象,对包含聚集和分组操作的聚集查询缺乏有效支持,因而难以用于优化海量数据库的查询性能.

本文以海量数据库应用为背景,就语义缓存技术如何拓展到海量数据库的聚集查询进行了深入研究,以面向聚集查询的语义缓存形式模型为基础,构造了海量数据库的语义缓存系统 StarCache.本文重点研究了查询处理、缓存管理和一致性维护等关键技术.本文研究所针对的海量数据库应用包含大量的 insert 操作,不包含 update 和 delete 操作,且海量数据库中的数据记录均包含时间信息.StarCache 已经集成在自主研发的并行数据库中间件 StarTP 之中,并在一项大型国家工程中得到了实际应用.

1 面向聚集查询的语义缓存模型

聚集查询的结果通常是一条或多条满足该聚集查询条件的记录.这些记录及其相关描述组成了该聚集查询产生的语义缓存项.语义缓存为一组语义缓存项的集合.由于语义缓存项完全由聚集查询语句决定,故本文用聚集查询语句作为语义缓存项的描述.本文讨论的聚集查询 SQL 语句由 select 子句、from 子句、where 子句和 group by 子句组成.其中:select 子句包含由非聚集属性构成的集合 A 以及聚集函数集合 F,select 子句中的集合 A 与 group by 子句的分组属性集合相同;from 子句中的关系集合记为 T;where 子句的条件谓词记为 P;聚集查询的结果记为 C.若将查询结果视为一个关系,则集合 A 中的每个属性和集合 F 中的每个聚集函数相当于关系中的某个属性.

本文的讨论以扩展的关系代数为基础,不仅支持投影、选择、连接操作,而且支持聚集和分组操作.不失一般性,假设关系的属性类型为实数域且忽略≠比较.现给出如下约定:

(1) 比较谓词 P 形式上写为 $X\ op\ c$,X 为关系属性,$op\in\{\leq,\geq,<,>,=\}$,c 表示常量.

(2) 连接谓词 P 形式上写为 $X=Y$,X 与 Y 为连接属性.

(3) 简单谓词:比较谓词和连接谓词的合称.

定义 **1**(语义缓存项). 给定数据库 $D=\{R_1,R_2,\ldots,R_m\}$,R_m 表示关系,语义缓存项 S 为五元组$\langle A,F,T,P,C\rangle$,其中:$T=\{R_{ik}|R_{ik}\in D,1\leq k\leq n\}$;$A=\{a_l|a_l$ 为 R_k 的属性,$R_k\in D\}$;$F=\{f(b)|b$ 为 R_s 的属性,$R_s\in T,f\in Ag\}$,$Ag=\{$MAX,MIN,SUM,COUNT$\}$;条件谓词 $P=p_1\wedge p_2\wedge\ldots\wedge p_j$,$p_j$ 为简单谓词;$C=\pi_{A,F}(\sigma_P(R_{i1}\times R_{i2}\times\ldots\times R_{in}))$,$R_{in}\in T$.语义缓存项中 C 的运算过程大致是:首先,产生 T 中关系的笛卡尔乘积;然后,利用条件 P 进行连接和选择;接着,对 A 的属性进行分组运算,计算 F 的聚集函数.Ag 本应包含 AVG,但由于 AVG 可转换为对聚集函数 SUM 和 COUNT 的计算,故本文不考虑 AVG 的处理.

下面举例说明定义 1.

例 1:假定数据库 D:*Product*(*Pid*,*Pname*,*Class*)和 *Log*(*Sn*,*Pid*,*Rank*,*Price*).语义缓存项 S 由如下查询产生:

Select *Pname*,*Rank*,*Count*(*Pname*) From *Product*,*Log* Where *Rank*<3 and *Product.Pid=Log.Pid* Group by

Pname,*Rank*;

语义缓存项 S 表示为$\langle A,F,T,P,C\rangle$,A={*Pname*,*Rank*},F={*Count*(*Pname*)},T={*Product*,*Log*},P=(*Rank*<3)∧(*Product*.*Pid*=*Log*.*Pid*),C 为聚集查询的结果.

本文按照定义 1 的方法描述聚集查询,例如,聚集查询 Q 可定义为$\langle A_Q,F_Q,T_Q,P_Q,C_Q\rangle$,其中每个组成部分的意义与定义 1 相同.查询 Q 与语义缓存项 S 的区别在于,C_Q 是虚拟的查询结果,而 C 是已经执行查询的结果.

为了利用缓存处理查询,首先需要从缓存中找出可以重用的缓存项,这一过程称为查询匹配.查询与缓存匹配的类型分为包含匹配、相交匹配和无关匹配:包含匹配是指查询可从缓存得到完整的结果;相交匹配是指可从缓存得到部分结果;查询不能从缓存中得到查询结果即为无关匹配.在按照定义 1 描述缓存项和查询之后,我们通过比较缓存项和查询对应的五元组来完成查询匹配.

聚集函数是聚集查询与 SPJ 查询的区别之一.聚集查询匹配判断必须讨论查询和语义缓存项匹配与聚集函数的关系.分组运算限定只有聚集查询的分组属性集合包含于语义缓存项的分组属性集合,二者才有可能匹配.在此条件下,查询与语义缓存项匹配时,它们的聚集函数之间存在多种可能的关系,二者的聚集函数集合完全等价是最直观的一种情况.除此之外,查询的聚集函数还可以基于语义,由语义缓存项的聚集函数和分组属性计算导出.

定义 **2**(聚集函数与语义缓存项的导出关系). 给定语义缓存项 $S=\langle A,F,T,P,C\rangle$,查询 $Q=\langle A_Q,F_Q,T_Q,P_Q,C_Q\rangle$.当 $f_1(a)\in F_Q$ 时,如果存在 $f_2(b)\in F$,使得以下 4 个条件之一成立,则称查询 Q 的聚集函数 $f_1(a)$可由 S 的 F 和 A 导出,记为 $f_1(a)=h(F,A)$.

(1) $f_1=f_2,a=b,f_1\neq$COUNT;

(2) $f_1=f_2,f_1=$COUNT;

(3) $f_1=$SUM,$f_2=$COUNT,$a\in A$;

(4) $f_1=$MAX|MIN,$a\in A$.

如果存在 $f_1(a)\in F_Q$,满足 $f_1(a)=h(F,A)$,则称查询 Q 的 F_Q 可由 S 的 F 和 A 导出,用 $F_Q\leftarrow D(F,A)$表示.对于任意 $f_1(a)\in F_Q$,满足 $f_1(a)=h(F,A)$,则称查询 Q 的 F_Q 可由 S 的 F 和 A 完全导出,用 $F_Q=D(F,A)$表示.如果 $F_Q\leftarrow D(F,A)$,存在 $f_1(a)\in F_Q$,不满足 $f_1(a)=h(F,A)$,则称查询 Q 的 F_Q 可由 S 的 F 和 A 部分导出,用 $F_Q\approx D(F,A)$表示.

查询与缓存项的匹配判断还需要考虑二者的条件谓词之间的相交和包含关系以及其他描述信息.

定义 **3**(查询与缓存项的无关匹配(**disjoint match**)和包含匹配(**containing match**)). 给定语义缓存项 $S=\langle A,F,T,P,C\rangle$,查询 $Q=\langle A_Q,F_Q,T_Q,P_Q,C_Q\rangle$,$P_A$ 为出现于 P_Q 中的属性集合:(1) 如果 $T=T_Q,A_Q\subseteq A,F_Q\leftarrow D(F,A),P_A\subseteq A$, $P\wedge P_Q$ 可满足,则 S 与 Q 匹配,否则 S 与 Q 无关匹配;(2) 如果 $T=T_Q,A_Q\subseteq A,F_Q=D(F,A),P_A\subseteq A,P_Q\Rightarrow P$,则 Q 完全由 S 包含,即包含匹配.

定义 **4**(查询与缓存项的相交匹配(**overlapping match**)). 给定语义缓存项 $S=\langle A,F,T,P,C\rangle$,查询 $Q=\langle A_Q,F_Q,T_Q,P_Q,C_Q\rangle$,$P_A$ 为出现于 P_Q 中的属性集合:如果 $T=T_Q,A_Q\subseteq A,P_A\subseteq A$,若以下 3 个条件之一成立,则查询 Q 与缓存项 S 相交匹配:(1) $F_Q=D(F,A),P_Q\Rightarrow P$ 不成立,$P\wedge P_Q$ 可满足;(2) $F_Q\approx D(F,A),P_Q\Rightarrow P$;(3) $F_Q\approx D(F,A),P_Q\Rightarrow P$ 不成立,$P\wedge P_Q$ 可满足.其中,条件(1)~条件(3)成立时的相交匹配依次称为条件相交、聚集函数相交和混合相交.

例 2:以例 1 中数据库与语义缓存项 S 为前提,观察如下 5 个查询:

Q_1:Select *Pname*,*Rank*,*Count*(*Pname*) From *Product*,*Log* Where *Rank*<2 and *Product*.*Pid*=*Log*.*Pid* Group by *Pname*,*Rank*;

Q_2:Select *Pname*,*Rank*,*Count*(*Pname*) From *Product*,*Log* Where *Rank*<4 and *Product*.*Pid*=*Log*.*Pid* Group by *Pname*,*Rank*;

Q_3:Select *Pname*,*Rank*,*Count*(*Pname*),*Sum*(*Price*) From *Product*,*Log* Where *Rank*<3 and *Product*.*Pid*= *Log*.*Pid* Group by *Pname*,*Rank*;

Q_4:Select *Pname*,*Rank*,*Count*(*Pname*),*Sum*(*Price*) From *Product*,*Log* Where *Rank*<4 and *Product*.*Pid*= *Log*.*Pid* Group by *Pname*,*Rank*;

Q_5:Select *Pname*,*Rank*,*Count*(*Pname*) From *Product*,*Log* Where *Rank*>3 and *Product*.*Pid*=*Log*.*Pid* Group by *Pname*,*Rank*.

显然,Q_1 包含于语义缓存项 S;Q_2,Q_3 和 Q_4 与语义缓存项 S 相交匹配,分别为条件相交、聚集函数相交和混合相交;Q_5 与语义缓存项 S 无关.

2 StarCache 关键技术

2.1 查询处理

查询匹配和查询分解是查询处理的重要步骤.查询匹配从缓存中找出能够为查询提供查询结果的缓存项.系统根据查询匹配结果确定是否分解查询,然后进行查询变换,最后完成查询处理.查询变换后得到的对缓存的子查询为探测查询(probe query),对后端数据库的查询为剩余查询(remainder query)[5].

对于无关匹配,将查询 Q 发送给服务器进行处理.对于包含匹配,将查询变换为对缓存的查询之后从缓存提取结果.当 Q 完全由语义缓存项 S 包含时,查询 Q 变换为探测查询 $Q_P=\langle A_Q,F',C,P\wedge P_Q,C_Q\rangle$,$F'$为 F_Q 映射到缓存项 S 的聚集函数集合.由定义 2 可知,F'由聚集函数 $f(x)$组成,$f(x)$由以下方法得到:若 $f_1(a)\in F_Q$,则存在 $f_2(b)\in F$,将 $f_2(b)$对应的属性列记为 f_2_b.

(1) 若 $f_1=f_2,a=b,f_1\neq$COUNT,则 $x=f_2\_b,f(x)=f_1(f_2\_b)$;

(2) 若 $f_1=f_2,f_1=$COUNT,则 $x=f_2_b,f(x)=$SUM(f_2_b);

(3) 若 $f_1=$SUM,$f_2=$COUNT,$a\in A$,则 $x=f_2_b,f(x)=a\times$SUM(f_2_b);

(4) 若 $f_1=$MAX|MIN,$a\in A$,则 $x=a,f(x)=f_1(a)$.

在相交匹配时,查询处理有两种可选策略:第 1 种策略是忽略相交缓存项,将查询发送给服务器处理;第 2 种策略是将查询分解,利用缓存处理检测查询而获得部分查询结果,将剩余查询发送给服务器处理,最后合并查询结果.这两种策略的选择依赖于相交匹配的具体情况.定义 4 给出了聚集查询 Q 与缓存项 S 相交匹配的 3 种类型.与此对应,查询 Q 可按照如下方式分解:

(1) 条件相交:探测查询 $Q_P=\langle A_Q,F',C,P\wedge P_Q,C_P\rangle$和剩余查询 $Q_R=\langle A_Q,F_Q,T_Q,\sim P\wedge P_Q,C_R\rangle$.

(2) 聚集函数相交:探测查询 $Q_P=\langle A_Q,F',C,P\wedge P_Q,C_P\rangle$和剩余查询 $Q_R=\langle A_Q,F_R,T_Q,P_Q,C_R\rangle$.

(3) 混合相交:探测查询 $Q_P=\langle A_Q,F',C,P\wedge P_Q,C_P\rangle$ 和剩余查询 $Q_{R1}=\langle A_Q,F_R,T_Q,P_Q,C_{R1}\rangle$,$Q_{R2}=\langle A_Q,F_Q,T_Q,\sim P\wedge P_Q,C_{R2}\rangle$.

聚集查询可以从查询条件和聚集函数两个维度分解,查询条件的分解是显而易见的,聚集函数的分解得到两个聚集函数集合 F'和 F_R.F'与包含匹配时的产生方法相同.$F_R=F_Q-\{f_1(a)|f_1(a)\in F_Q,f_1(a)=h(F,A)\}$,表示通过删除 F_Q 中可由 F 和 A 导出的聚集函数得到剩余查询的函数集合.

分解方式(2)和分解方式(3)的剩余查询都需要处理与查询 Q 相同的数据记录,与直接将查询 Q 发送给服务器处理相比没有优势.当发生聚集函数相交或者混合相交时,我们选择将查询 Q 发送给服务器处理的方式.对于条件相交的情况,则采用查询分解策略处理.

假定待处理聚集查询为 Q,语义缓存中缓存项为 $S^1,S^2,\ldots,S^n$(n 为正整数).基于上述查询处理分析以及定义 3 和定义 4,下面给出基于缓存的聚集查询处理 AQP(aggregate query processing)算法.

算法 **1**. AQP 算法.

```
L1  m_type=0;   //置匹配类型 m_type 为无关匹配
L2  for (i=1;i<=n;i++) {
L3    if (S^i.T=T_Q and A_Q⊆S^i.A) then {
L4    if (P_Q⇒S^i.P) then {
L5    If (F_Q=D(S^i.F,S^i.A)) then {m_type=4; Sm=S^i; break}   //包含匹配
L6    Else {if (m_type=0 and F_Q≈D(S^i.F,S^i.A)) then m_type=1;}   //聚集函数相交
L7    } Else if (S^i.P∧P_Q 可满足) then {
```

L8 If (m_type=0 and $F_Q \approx D(S^i.F,S^i.A)$) then m_type=2; //混合相交

L9 else if (m_type<3 and $F_Q=D(S^i.F,S^i.A)$) then {m_type=3;$Sm=S^i$;}}} //条件相交匹配

L10 修改 Sm 的访问记录;

L11 *switch*(m_type) {

L12 Case 0:启动缓存替换机制; //参见第 3.2 节

L13 Case 1,Case 2:发送查询 Q 到数据库处理;break;

L14 Case 3:分解查询为 Q_P 和 Q_R;

L15 基于缓存项 Sm 处理 Q_P,同时将 Q_R 发送到数据库处理;

L16 汇总查询结果;break;

L17 Case 4:将查询 Q 变换为 Q_P,基于缓存项 Sm 处理 Q_P;

L18 }

L2~L10 比较聚集查询 Q 与 n 个缓存项,考虑了包含匹配、相交匹配和无关匹配 3 种情况.L11~L16 根据查询匹配的类型处理查询,其中:L14~L16 将查询按照条件分解为等价的子查询进行处理,保证了查询结果与原始查询相同;其余几种情况不需要分解查询也能够保证查询结果的正确性.按照最坏情况考虑,AQP 算法在聚集查询 Q 与所有缓存项进行比较之后才能得到最终处理结果,缓存项总数为 n,因此算法复杂度为 $O(n)$.

2.2 缓存替换管理

为了便于管理,本文将缓存分为持久缓存、易失缓存和虚拟缓存 3 种类型.持久缓存由应用通过显式命令创建,不参与缓存替换过程;虚拟缓存一般只保存不包含实际数据的缓存项,用于缓存替换管理,不参与查询处理的匹配;易失缓存是基于查询局部性形成的缓存项集合,由虚拟缓存转换得到.缓存的类型可以根据条件转换,转换时机包括创建缓存、删除缓存和缓存替换等.持久缓存项在被用户删除的时候转换为虚拟缓存项.如果用户创建缓存项时存在与其匹配的易失缓存项,那么将它们合并成一个持久缓存项.根据缓存替换策略,易失缓存中被淘汰出去的项目自动转为虚拟缓存,虚拟缓存中被选中的缓存项进入易失缓存.持久缓存项和易失缓存项在转换为虚拟缓存时,它们所包含的查询结果将被继续保留,直到最终被淘汰为止.当它们再次转变为易失缓存时,系统可以直接利用已有的缓存数据,减少查询开销.

缓存替换的常见算法有 LRU(least recently used)和 LFU(least frequently used)以及相关衍生算法.LRU 强调最近访问的时间,LFU 突出访问的频率[17].我们结合访问时间和频率,提出了 LFURC(least frequently used in recent cycle)算法,基于最近周期内访问次数进行缓存替换.语义缓存与查询存在无关、相交和包含匹配 3 种关系,查询对语义缓存项的引用不能按照传统的缓存管理方法简单计数.因此,LFURC 为无关匹配、相交匹配和包含匹配指定了不同的访问系数:无关匹配对应访问系数为 0,相交匹配对应访问系数为 0.5,包含匹配对应访问系数为 1.当用户提交的查询与语义缓存项匹配时,系统修改匹配的缓存项引用历史,记录匹配的类型和时间.若用户提交的查询与语义缓存无关匹配,则该查询可能会参与虚拟缓存和易失缓存的替换.替换过程需要计算该查询在虚拟缓存中对应缓存项的替换值、其他虚拟缓存项的替换值和易失缓存项的替换值.假定语义缓存项 S 在最近 Tr 时间内共被访问 m 次,第 i 次访问系数为 λ_i,语义缓存项 S 的替换值为 $V_R=\sum_{i=1}^{m}\lambda_i$.聚集查询结果占用的空间与被查询的关系相比要小得多,相对而言,缓存维护的开销更值得关注,因而我们以缓存项的个数作为衡量缓存替换的标准.

假定 St 为易失缓存,包含$|St|$个缓存项,缓存项个数限制为 Nt;Sv 为虚拟缓存,包含$|Sv|$个缓存项,缓存项个数限制为 Nv,待处理聚集查询为 Q_t.下面给出 LFURC 算法.

算法 **2**. LFURC 算法.

L1 m_type=0

L2 for (i=1;i<=$|Sv|$;i++) {

L3 比较 Q_t 和 Sv^i; //参见查询匹配定义

L4 if (Q_t包含于 Sv^i) then {m_type=1;
L5 修改 Sv^i 访问记录;
L6 Sv^m=Sv^i;
L7 break;
L8 }
L9 if (Q_t与 Sv^i 相交匹配) then 修改 Sv^i 访问记录;
L10 }
L11 if (m_type=0) //Q_t不包含于虚拟缓存
L12 then { if |Sv|<Nv then $Sv.add(Q_t)$; //将 Q_t加入 Sv
L13 else {
L14 计算 Sv 中所有缓存项的 V_R值;
L15 找出 Sv 中具有最小 V_R值的 Sv^k;
L16 $Sv.remove(Sv^k)$; //将 Sv^k从 Sv 中删除
L17 $Sv.add(Q_t)$; //将 Q_t加入 Sv
L18 }}
L19 else { //进入易失缓存的替换处理
L20 if |St|<Nt then {$St.add(Sv^m)$;$Sv.remove(Sv^m)$;} //若易失缓存空间足够,则将 Sv^m加入 St
L21 else {
L22 计算 St 中缓存项的 V_R值;
L23 找出 St 中具有最小 V_R值的缓存项 St^l;
L24 if ($St^l.V_R$<$Sv^m.V_R$) then{
L25 $St.remove(St^l)$; //将 St^l从 St 中删除
L26 $Sv.add(St^l)$ //将 St^l添加到 Sv
L27 $Sv.remove(Sv^m)$; //将 Sv^m从 Sv 中删除
L28 $St.add(Sv^m)$ //将 Sv^m添加到 St
L29 }}}

缓存替换算法分为两个阶段:L1~L18 实施对虚拟缓存的替换管理;L19~L28 实施对易失缓存的替换管理.虚拟缓存的替换管理使得只有访问频率达到一定频率的聚集查询才可能进入易失缓存,参与易失缓存替换.若替换算法经过第 1 阶段处理就结束,则算法复杂度为 $O(|Sv|)$;若替换经过两个阶段才处理完毕,则算法复杂度为 $O(|Sv|+|St|)$.与传统的缓存替换算法相比,LFURC 的开销略大,但由于语义缓存中缓存项数目比传统缓存中缓存项数目要小,因此 LFURC 的相对开销并不大.

2.3 一致性维护

本文的一致性维护方法是针对海量数据库应用包含大量的 insert 操作,不包含 update 和 delete 操作,且海量数据库中的数据记录均包含时间信息的情况而提出的.因此,每个语义缓存项可视为若干连续时间段内数据的查询结果,语义缓存项的数据由对这些时间段的查询结果汇总得到.基于这种划分方法,一致性维护的工作可转化为定时查询最近一段时间的数据,将结果添加到语义缓存项中.为此,我们提出定时增量维护算法 PIM (periodical incremental maintenance).假定语义缓存中缓存项为 $S^1,S^2,\ldots,S^n$(n 为正整数),给出 PIM 算法如下:

算法 **3**. PIM 算法.

L1 for (i=1;i<=n;i++) {
L2 取当前时间 t_n;
L3 t=t_n–$S^i.t_m$; //计算当前时间与上次维护之间的时间差
L4 if (t>=$S^i.T$) then { //若维护时间差已经超过 S^i的维护周期 $S^i.T$,则开始维护

L5 根据 S^i 的描述信息构造查询时间区间($S^i.t_m$,$S^i.t_m+S^i.T$]之内数据的语句 Q_m;

L6 启动线程,将查询 Q_m 传递给线程,执行 L7~L9;

L7 将查询语句 Q_m 提交给数据库处理;

L8 将 Q_m 的结果加入 S^i;

L9 $S^i.t_m=S^i.t_m+S^i.T$; //更新 S^i 的维护时间戳

L10}}

L2~L5 检查缓存项是否需要更新,并重构更新查询语句;L6~L9 以多线程方式并发执行缓存项的更新查询.算法 PIM 利用应用仅包含 insert 操作这一特点,以增量方式将对新数据的查询结果并入缓存中.若将维护周期作为应用能够容忍的数据偏差时间,这种方法就保证了缓存项的弱一致性.由于缓存包含 n 个待维护的缓存项,若将对每个缓存项的维护视为一个整体操作,则算法复杂度为 $O(n)$.

定时增量维护的时机与每个语义缓存项相关,每个语义缓存项对应的聚集查询对于一致性维护有着不同的需要.如果聚集查询面向的业务有较短的周期,那么定时维护的周期也应该相应缩短.因此,为了及时更新语义缓存数据,缓存系统给定一个默认的维护周期,以这个周期定时调用 PIM 算法,使其能够满足基本的更新需要,同时系统也支持应用定制维护周期.

3 实现及测试

3.1 实 现

我们在并行数据库中间件 StarTP 上实现了面向聚集查询的中间层语义缓存 StarCache.StarTP 是基于 CORBA(common object request broker architecture)[18]的并行数据库中间件,支持并行加载和并行查询.它已经成功应用于多个海量数据库系统.StarTP 的结构如图 1 所示,包括如下组件:(1) 语法分析器(syntax parser):分析用户提交的 SQL 语句,生成语法树;(2) 数据字典(data dictionary):保存数据库集群中表的分布状况等信息;(3) 数据划分器(data partition):选择数据记录发送的目的数据库节点;(4) 表加载器(table loader):将数据记录写入特定数据库节点;(5) 并行查询(parallel query):生成并行查询规划,分发子查询和汇总查询结果;(6) 中间层语义缓存 StarCache:优化聚集查询;(7) 查询对象(query object):负责对特定数据库节点的查询.其中:数据划分器和表加载器为并行加载服务的组成部分;并行查询、语义缓存和查询对象构成并行查询服务;语法分析器和数据字典为并行加载和并行查询的辅助设施.

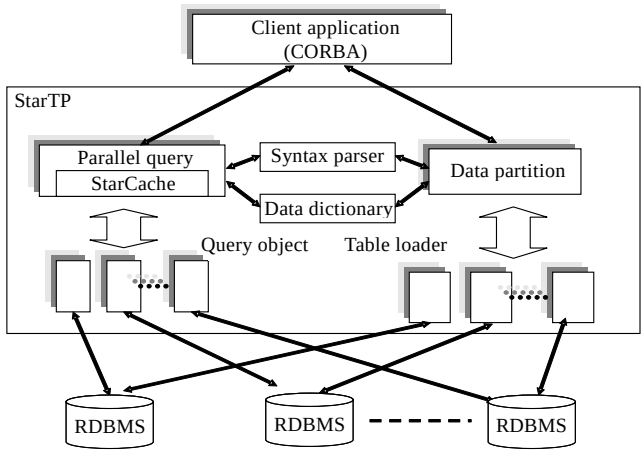

Fig.1 Architecture of StarTP

图 1 StarTP 体系结构

StarCache 的结构如图 2 所示,其中各组成部分如下:

(1) 查询接口(query interface,简称 QI):接收用户的查询请求以及定制缓存的请求.

(2) 查询处理器(query evaluator,简称 QE):完成基于缓存的查询处理.

(3) 查询匹配(query matching,简称 QM):确定语义缓存是否可满足查询.

(4) 元数据表(metadata table,简称 MT):存储语义缓存中缓存项的描述、引用等信息,其中每条记录与语义缓存中缓存项一一对应.

(5) 缓存(cache store,简称 CS):存储语义缓存的数据,即缓存的查询结果.StarCache 的缓存由中间层的数据库服务器实现.

(6) 缓存替换(cache replacement,简称 CR):当查询处理确定缓存替换的时机以及选择进入缓存的查询和淘汰的缓存时,实现了 LRU,LFU 和 LFURC 算法,支持用户配置策略.

(7) 一致性管理器(consistency manager,简称 CM):保持语义缓存的数据与后端数据库一致,避免从缓存中取得过时的数据,保证查询处理结果的正确性.StarCache 的实现不需要修改数据库的内核.StarCache 通过调用对数据库的接口,实现对后端数据库的查询和缓存数据库的查询.StarCache 的一致性管理器将一致性维护的查询语句提交给后端数据库执行,将更新缓存项的语句提交给中间层的数据库服务器执行.

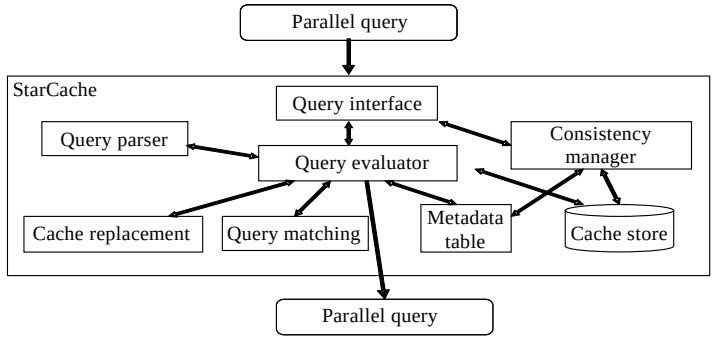

Fig.2 Architecture of StarCache

图 2 StarCache 体系结构

StarCache 的工作机制如图 3 所示.并行查询 PQ 提交聚集查询语句给 QI(1),QI 将查询转交给 QE 处理(2).QE 首先调用 QP 分析查询语句(3),通过 QM 确定是否需要利用缓存处理查询(4).QM 比较查询与 MT 的缓存信息,确定匹配的类型(5).然后,QE 根据匹配结果进行处理:若缓存能够为查询提供全部结果,则变换查询,基于缓存处理查询(6);若缓存不能满足查询,则直接交给 PQ 查询数据库(7).为了充分利用缓存资源,在缓存只能为查询提供部分结果时将查询分解为两部分:一部分基于缓存处理(6);另一部分基于数据库处理(7).QE 将(6)和(7)的结果合并,得到最终查询结果.在处理(6)、(7)的时候,QE 更新 MT 中缓存项的信息(8).当缓存不能满足查询时,与(7)同时启动缓存替换机制,确定是否需要缓存新的查询,从而保证缓存适应查询请求的变化(9).CR 依据 MT 的缓存引用信息(10),按照替换策略调整 CS 的内容(11).在语义缓存启用时,QI 启动 CM,由 CM 定期维护缓存一致性(12).CM 查找 MT,确定需要维护的语义缓存项(13),通过 PQ 查询数据库取得最新数据变化(14),更新 CS(15).

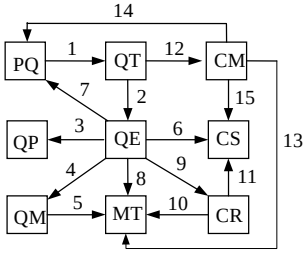

Fig.3 Mechanism of StarCache

图 3 StarCache 工作机制

3.2 性能测试

实验环境由 5 台数据库服务器、1 台应用服务器和 5 台普通 PC 组成.每台服务器配置为两个 Intel Xeon 2GHz CPU,4G RAM,两个 70GB 的 SCSI 硬盘.服务器操作系统为 Redhat Linux7.3,数据库服务器为 Oracle8.1.7.PC 机的配置为 P4 1.6GHz,256 兆内存,操作系统为 Win2000 Professional.所有机器通过 100Mbps 以太网互联.

数据库的数据来源于 StarTP 应用中的业务数据.测试使用的查询为实际应用中的常用查询.

实验 1 测试语义缓存对聚集查询性能的影响,测试方法为对比同一数据库在不同数据规模下有缓存和无缓存时的查询响应时间.在实际应用中,聚集查询中的分组属性具有有限取值,在此基础上形成的缓存包括的记录相对于数据库中需要处理的记录要少得多,因此,基于缓存处理查询能够明显改善查询性能.图 4 是一组聚集查询在有缓存和无缓存情况下的测试结果.图 4 说明,有缓存和无缓存时的聚集查询响应时间有较大差距,语义缓存提高了聚集查询的性能.

实验 2 对比测试语义缓存和物化视图对聚集查询的优化作用.由于 Oracle 的物化视图对异地关系有所限制,在单节点环境下,语义缓存和物化视图的可比性更好.因此,实验 2 配置后端数据库为单节点.实验 2 以相同查询流输入,语义缓存只保留持久缓存,按照物化视图与持久缓存中缓存项一一对应的方式构造缓存和物化视图.图 5 说明 StarCache 的查询处理性能与物化视图相当.它们的性能略有差异是由于物化视图在数据库本地处理,而 StarCache 在处理查询时存在网络延时和中间层处理开销,在查询时间比较长时可以忽略.

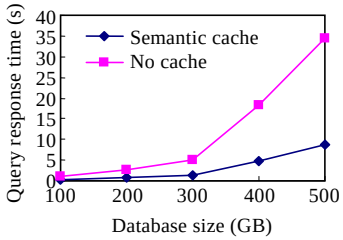

Fig.4 Impact of semantic cache on aggregate queries

图 4 语义缓存对聚集查询的作用

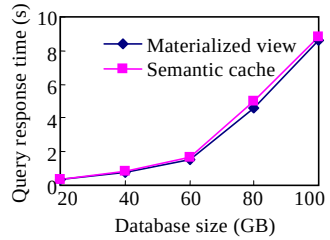

Fig.5 Performance comparison between semantic cache and materialized view

图 5 语义缓存与物化视图的性能对比

实验 3 测试查询匹配开销、平均查询响应时间与语义缓存规模的关系.实验 3 以相同的聚集查询集合为输入,基于语义缓存处理聚集查询,考察语义缓存规模增大时查询匹配开销与查询响应时间之间的关系.从表 1 可以看到,随着语义缓存中语义缓存项目的增多,查询匹配的开销逐渐增大,但是查询匹配的开销占查询响应时间的百分比仍然较低.因为海量数据库应用的聚集查询响应时间一般比较长,所以查询匹配的开销与语义缓存带来的性能改善相比是无关紧要的.

Table 1 Aggregate query matching cost and query response time with different size of semantic cache

表 1 不同语义缓存规模下的聚集查询匹配开销和查询响应时间

Number of semantic cache items	Query response time (ms)	Query matching cost (ms)	Query matching cost/ query response time (%)
50	187.677	0.875 86	0.467
100	188.897	1.751	0.927
150	189.235	2.631	1.389
200	190.077	3.501	1.842
250	192.453	4.368	2.269

实验 4 测试语义缓存规模对聚集查询匹配的影响.实验 4 以相同的聚集查询集合为输入,基于语义缓存处理聚集查询,考察不同语义缓存规模下包含匹配、相交匹配和无关匹配的分布.图 6 表明,在相同条件下,随着语义缓存规模的增大,聚集查询与缓存项匹配成功的几率也逐渐增大.

实验 5 测试语义缓存替换策略对查询匹配命中率的影响,包括 LRU,LFU 和 LFURC 策略.这里的命中包括了相交匹配和包含匹配.图 7 表明 LFURC 比 LRU 和 LFU 具有更好的性能.

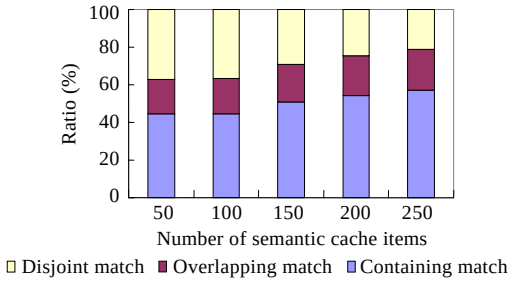

Fig.6 Ratio of aggregate query matching

图 6 聚集查询匹配类型比例

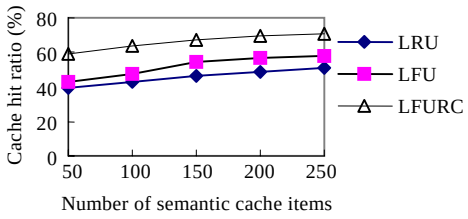

Fig.7 Performance comparison of replace policies

图 7 替换策略的性能比较

4 结束语

聚集查询是在海量数据库应用中常见的查询类型,提高这类查询的处理效率是海量数据库应用所面临的重要挑战.语义缓存为优化查询性能提供了有效途径,然而,当前的语义缓存研究以 SPJ 查询为重点,缺乏对聚集查询的深入研究.为了能够在海量数据库应用环境下利用语义缓存优化聚集查询,我们扩展了现有的语义缓存研究,提出并实现了面向聚集查询的中间层语义缓存系统 StarCache.

本文的研究成果已投入一项国家大型工程应用,该应用的数据规模达到了 TB 级,其查询业务包含大量的聚集查询.实践表明,本文提出并实现的 StarCache 缓存系统是有效的.进一步的工作包括优化更为复杂的聚集查询和基于知识库的聚集查询匹配等.

References:

[1] Zaharioudakis M, Cochrane R, Lapis G, Pirahesh H, Urata M. Answering complex SQL queries using automatic summary tables. In: Dunham M, Naughton JF, Chen W, Koudas N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2000. 105−116.

[2] Bello RG, Dias K, Downing A, Feenan J, Finnerty J, Norcott WD, Sun H, Witkowski A, Ziauddin M. Materialized views in oracle. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th Int'l Conf. on VLDB. San Fransisco: Morgan Kaufmann Publishers, 1998. 659−664.

[3] Goldstein J, Larson P. Optimizing queries using materialized views: A practical, scalable solution. In: Sellis T, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2001. 331−342.

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

[5] Dar S, Franklin MJ, Jonsson BT, Srivastava D, Tan M. Semantic data caching and replacement. In: Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL, eds. Proc. of the 22nd Int'l Conf. on Very Large Data Base. San Fransisco: Morgan Kaufmann Publishers, 1996. 330−341.

[6] Ren Q, Dunham MH, Kumar V. Semantic caching and query processing. IEEE Trans. on Knowledge and Data Engineering, 2003, 15(1):192−210.

[7] Basu J. Associative caching in client-server databases [Ph.D. Thesis]. Stanford: Stanford University, 1998.

[8] Amiri K, Park S, Tewari R, Padmanabhan S. DBProxy: A dynamic data cache for Web applications. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering (ICDE 2003). Washington: IEEE Computer Society, 2003. 821−831.

[9] Soundararajan G, Amza C. Using semantic information to improve transparent query caching for dynamic content Web sites. In: Cantarella JD, ed. Proc. of the Int'l Workshop on Data Engineering Issues in E-Commerce (DEEC 2005). Washington: IEEE Computer Society, 2005. 132−138.

[10] Chen L. Semantic caching for XML queries [Ph.D. Thesis]. Worcester: Worcester Polytechnic Institute, 2004.

[11] Ren Q, Dunham MH. Using clustering for effective management of a semantic cache in mobile computing. In: Banerjee S, Chrysanthis PK, Pitoura E, eds. Proc. of the Int'l Workshop on Data Engineering for Wireless and Mobile Access. New York:

ACM Press, 1999. 94−101.

[12] Ren Q, Dunham MH. Using semantic caching to manage location dependent data in mobile computing. In: Pickholtz R, Das SK, Caceres R, Garcia-Luna-Aceves JJ, eds. Proc. of the 6th Annual Int'l Conf. on Mobile Computing and Networking (MOBICOM 2000). New York: ACM Press, 2000. 210−221.

[13] Wu TT. The research on client semantic caching in mobile database [Ph.D. Thesis]. Changsha: National University of Defense Technology, 2002 (in Chinese with English abstract).

[14] Lee K, Leong HV, Si A. Semantic query caching in a mobile environment. ACM SIGMOBILE Mobile Computing and Communications Review, 1999,3(2):28−36.

[15] Kalnis P, NG WS, Ooi BC, Papadias D, Tan K. An adaptive peer-to-peer network for distributed caching of OLAP results. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2002. 25−36.

[16] Garbacki P, Epema D, Van Steen M. A two-level semantic caching scheme for super-peer networks. In: Biersack E, Rodriguez P, eds. Proc. of the IEEE 10th Int'l Workshop on Web Content Caching and Distribution (WCW). Washington: IEEE Computer Society, 2005. 47−55.

[17] Lee D, Choi J, Kim JH, Noh SH, Min SL, Cho Y, Kim CS. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. on Computers, 2001,50(12):1352−1361.

[18] Object Management Group. The common object request broker: architecture and specification, version 3.0.3, 2004. http://www.omg.org/technology/documents

附中文参考文献:

[13] 吴婷婷.移动数据库客户语义缓存的研究[博士学位论文].长沙:国防科学技术大学,2002.

蔡建宇(1976－),男,湖南宁远人,博士,主要研究领域为数据库,分布计算.

贾焰(1960－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,分布计算.

吴泉源(1942－),男,教授,博士生导师,主要研究领域为智能软件,分布计算.

邹鹏(1957－),男,教授,博士生导师,主要研究领域为操作系统,分布计算.