

# 基于大规模事务处理系统的中间层语义缓存技术的研究

向阳,杨树强,蔡建宇,贾焰  
(国防科技大学 计算机学院,湖南 长沙 410073)  
(adam80429@hotmail.com)

**摘要:**对当前的中间层语义缓存的研究进行了分析,以一个大规模事务处理系统为背景,提出了一个针对海量数据统计分析的中间层语义缓存解决方案,给出了其相关定义、体系结构及管理机制,并在大规模事务处理系统上对语义缓存进行了测试。

**关键词:**语义缓存;缓存管理;大规模事务处理系统;统计分析查询  
**中图分类号:** TP311.13 **文献标识码:** A

## Research of semantic caching based on large-scale transaction processing system

XIANG Yang, YANG Shu-qiang, CAI Jian-yu, JIA Yan

(College of Computer Science, National University of Defense Technology, Changsha Hunan 410073, China)

**Abstract:** The research on semantic cache in current middle layer was analyzed, and a favourable solution aiming to mass data statistics on semantic cache in middle layer was proposed. Furthermore, some definitions related, architecture and management mechanisms were given based on a large-scale transaction processing system on which semantic cache was tested.

**Key words:** semantic caching; cache manage; large-scale transaction process system; statistic query

### 0 引言

在海量数据库应用中,统计分析占很大一部分任务,其工作非常耗时,是影响系统性能的重要因素。在 OLTP(联机事务处理)应用中,这一因素更为突出。对统计分析查询进行优化,缩短查询响应时间就成为了提高海量数据库性能的主要方面。在众多的数据库优化方法中,缓存技术得到了业界的广泛关注。页缓存、元组缓存和语义缓存等技术相继出现,为提高数据库应用的性能发挥了巨大作用。特别是语义缓存技术弥补了页缓存和元组缓存不能很好支持关系数据库的缺陷,具有更广泛的意义。

语义缓存是将查询结果和相关语义信息缓存,基于查询谓词为将来的查询提供解答。现有的语义缓存技术大多数为客户端语义缓存或以移动环境为背景提出,它们通常都只能为单个用户提供服务,并且由于资源有限,缓存不可能占有太多空间,仅适应于客户请求比较简单固定的应用,很难提高缓存的利用率,对于当前海量数据库性能的提高作用有限。

大规模事务处理系统是一个典型的三层结构海量数据库应用系统,其主要目标是实现对数据库的海量数据插入和查询。大规模事务处理系统的所有功能由中间层应用服务器提供。根据对现有应用需求的分析,目前大规模事务处理系统处理的海量数据规模达到了 PB 级,统计类型的查询大约占查询任务的 1/3 以上。由于统计查询涉及的日志表包括较大数据量,统计查询的处理需要消耗大量的数据库服务器资源。随着业务的扩展,大规模事务处理系统处理的数据规模会进一步扩大。面向 TB 乃至 PB 级的数据库应用的统计分析需求,如何提高统计分析的性能就成为了大规模事务处理系统的一个急需解决的问题。大规模事务处理系统采用了三层体系结构,因此在中间层部署语义缓存,通过在中间层处理查询

是优化查询、提高系统性能的有效途径。

### 1 相关工作

本文立足于大规模事务处理系统,研究分析了在中间层实现语义缓存所需的功能组件,提出适用于中间层的语义缓存体系结构。语义缓存的组件包括语法分析器、查询匹配、查询剪裁与处理、缓存替换管理、一致性管理等功能部分和缓存元数据、缓存等存储部分。具体结构如图 1 所示。

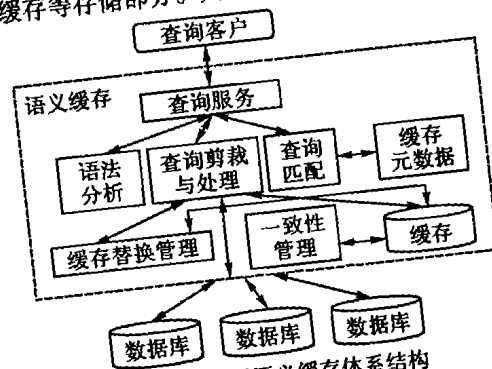


图1 中间层语义缓存体系结构

虽然当前语义缓存研究已经取得了不少成果,但是存在许多不足之处:1)语义缓存支持的查询类型还比较缺乏处理复杂查询的能力。如现有语义缓存大多以(Select Project Join)类型查询为研究对象;2)语义缓存的描述方法比较单一,还有待进一步的研究;3)语义缓存的管理和一致性维护也一直是研究的难点。本文目的是海量数据统计分析的特性,提供一个适合海量数据统计分析的中间层语义缓存解决方案。语义缓存主要包括两个部分:语义缓存管理和基于语义缓存的查询处理,本文仅讨论语义缓存管理部分。

收稿日期:2005-01-28

基金项目:国家 863 计划资助项目(2004AA112020;2003AA115210;2003AA111020)

作者简介:向阳(1980-),男,湖南人,硕士研究生,主要研究方向:分布计算、数据库;杨树强(1968-),男,湖南人,教授,主要研究方向:分布计算、数据库;蔡建宇(1976-),男,湖南人,博士研究生,主要研究方向:分布式计算、数据库;贾焰(1961-),女,四川人,博士生导师,主要研究方向:分布式计算、数据库。

## 2 语义缓存定义

### 2.1 可缓存查询

对于用户查询,只考虑带有统计特性的查询语句,即用户查询必须是可缓存查询。可缓存查询是指能够利用语义缓存获取结果的查询,其定义如下:

定义 1 如果查询语句  $Q$  同时满足以下条件:

- 1) 数据库关系集中含有日志表;
- 2) 投影属性集中含有聚合函数;
- 3) 分组属性有限取值;
- 4) 条件子句都是通过逻辑运算符 AND 连接的多个比较表达式。

则称语句  $Q$  为可缓存查询语句。

在定义 1 中,日志表是指对统计分析性能影响很大的数据库表,其数据量一般在几十 GB 级以上。

之所以仅仅缓存符合上述要求的语义查询,主要是出于以下两方面的考虑:1) 本文讨论的是针对海量数据的统计分析查询优化,所以仅对日志表进行缓存;2) 在统计分析查询中,如果是分组属性值域不是有限取值的,那么其分组之后结果集可能很大,对这种结果集进行缓存的意义不大。若无特别声明,下面讨论的查询都是指可缓存查询。

例 1 考虑全国人口信息资源数据库中的一个关系表 People(Pno, Pname, Age, Adress, registertime), 属性集中 Age 的值域是可数有限的,即  $0 < \text{Age} < 150$ , 那么对于年龄的统计分析就是可缓存查询。

```
Select Age, count(Age) a From People Where Adress = 'hunan' And
registertime > '2003-01-01' And registertime < '2004-01-01' Group By Age Order By a;
```

### 2.2 语义缓存定义

海量数据统计分析查询针对的表大多具有庞大的数据量。从一般统计查询的分析可知,统计往往是对具有有限取值的属性列实施,基于这样一些属性列进行粒度不同的分组统计可以实现不同类型的统计。统计时间是统计查询的基本要素,统计时间段之间的相关性是重用查询结果的基础之一,所以在语义缓存的定义中需要对统计属性列和统计时间进行特殊处理。下面给出语义缓存的定义。

假设  $D$  由一系列数据库关系  $R_1, R_2, \dots, R_n$  组成,即  $D = \{R_i, 1 \leq i \leq n\}$ ,  $A_{R_i}$  为  $R_i$  的属性集合。

定义 2 给定数据库  $D = \{R_i\}$ , 缓存语义项  $s$  是一个用  $\langle R, A, A', G, P, T \rangle$  表示的六元组,其中  $R \in D, A' \subseteq \{COUNT, SUM, MAX, MIN, AVG\}$ ;  $A, G \subseteq A_R$ ;  $P = P_1 \vee P_2 \dots \vee P_m$ , 其中  $P_j = b_{j1} \wedge b_{j2} \wedge \dots \wedge b_{jm}$ ,  $1 \leq j \leq m, b_{jk}$  是不含统计时间的条件谓词;  $T = T_1 \vee T_2 \vee \dots \vee T_m$ , 其中  $T_k = c_{k1} \wedge c_{k2} \wedge \dots \wedge c_{km}$ ,  $1 \leq k \leq m, c_{ki}$  是仅含统计时间的条件谓词。

在定义 2 中,  $R$  和  $A$  分别表示查询涉及的关系表和属性集合,  $A'$  是查询涉及的聚合函数集合,  $G$  是查询涉及的分组属性集合,  $P$  是不含统计时间的条件子句,  $T$  是仅含统计时间的条件子句,这几个元素表达了语义缓存的基本特征。在缓存语义项定义中,将分组属性集合和仅含统计时间的条件子句单独分离出来集中体现了海量数据统计分析的特性。

例 2 对例 1 中对年龄的统计分析查询,其语义缓存项  $S$  为  $\langle \text{People}, \text{Age}, \text{count}(\text{Age}), \text{Age}, \text{Adress} = 'hunan', \text{registertime} > '2003-01-01' \text{ and } \text{registertime} < '2004-01-01' \rangle$ 。

## 3 语义缓存管理

语义缓存的管理包括缓存的初始化、缓存的添加、缓存的删除和缓存的替换。当语义缓存不能满足用户查询时,为了让将来的用户查询能够重用此次查询结果,语义缓存就需要保存当前查询的结果及其语义信息。随着用户查询增多,语义缓存的内容可能会随之增多。当语义缓存占用空间达到一定限度,它就必须将相对作用较小的缓存数据淘汰出去,提高缓存空间利用率。语义缓存的替换就是在恰当的时机淘汰缓存中原有的数据,然后加入新的缓存数据。缓存的替换实际上是按照一定策略进行缓存添加和缓存删除。

### 3.1 缓存状态定义

为了便于管理,我们将语义缓存划分为三个部分,其定义如下:

定义 3

1) 强制缓存:用户自定义的语义缓存。在语义缓存启动的时候,通常用户可以预先自定义一组语义缓存项,实现语义缓存的初始化。这个过程将用户自定义缓存项加入到强制缓存中。

2) 临时缓存:在查询处理过程中按照一定策略确定需要而创建的缓存为临时缓存。

3) 虚拟缓存:当用户提交的可缓存查询语句不能从强制缓存和临时缓存中取得结果时,语义缓存系统将这条语句的描述信息存入虚拟缓存中。虚拟缓存中保存的仅仅为可缓存查询语句的信息,不是真正的语义缓存。虚拟缓存的作用是记录最近一段时间可缓存查询的访问次数,为虚拟缓存中的项目进入临时缓存提供判断依据。

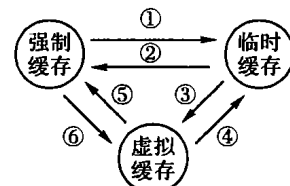


图2 缓存状态转换

### 3.2 缓存状态转换

强制缓存、临时缓存和虚拟缓存之间可以按照一定条件相互转换。图 2 为三种状态之间关系示意图。以下为三种状态之间的转换条件:

①指的是缓存项  $B$  从强制缓存状态转换至临时缓存状态,其转换时机为:用户删除该强制缓存项  $B$ , 但该缓存项  $B$  的使用较频繁,因此,还有使用价值,且符合临时缓存条件,则将该缓存项由强制缓存转换为临时缓存。

②指的是缓存项  $B$  从临时缓存状态转换至强制缓存状态,其转换时机为:用户创建一个强制缓存项  $B'$ , 且该缓存  $B$  与  $B'$  是可合并的,则需要创建一个由  $B$  和  $B'$  合并的强制缓存项,并将临时缓存项  $B$  移出临时缓存队列。

③指的是缓存项  $B$  从临时缓存状态转换至虚拟缓存状态,其转换时机为:通过替换策略须要替换掉临时缓存项  $B$ , 但该缓存项  $B$  符合虚拟缓存条件,则将该缓存项由临时缓存转换为虚拟缓存。

④指的是缓存项  $B$  从虚拟缓存状态转换至临时缓存状态,其转换时机为:用户对虚拟缓存项  $B$  的访问频繁,符合临时缓存条件,且应用临时缓存的替换策略,  $B$  不是替换对象,则将  $B$  由虚拟缓存队列移至临时缓存队列。

⑤指的是缓存项  $B$  从虚拟缓存状态转换至强制缓存状态,其转换时机为:用户创建一个强制缓存项  $B'$ , 且  $B$  与  $B'$  是可合并的,则需要创建一个由  $B$  和  $B'$  合并的强制缓存项,并将虚拟缓存项  $B$  移出虚拟缓存队列。

⑥指的是缓存项  $B$  从强制缓存状态转换至虚拟缓存状

态,其转换时机为:用户删除该强制缓存项 B,但该缓存项 B 满足虚拟缓存条件,则将该缓存项由强制缓存转换为虚拟缓存。

### 3.3 缓存替换策略

缓存替换主要的作用是控制缓存规模,将缓存的规模维持在一个合理的数量,从而提高缓存的性能。当有新的缓存项需要放到缓存,而缓存中的缓存项又达到一定数量时,缓存替换算法要选出缓存项进行替换,使缓存发挥最大的作用。

基于上述三种缓存状态,本文提出一个 WLFU (Weighted LFU) 算法实现缓存替换。基本思想是为临时缓存和虚拟缓存中每个项目记录在最近的一段时间之内的访问次数,当用户查询与临时缓存或者虚拟缓存中项目匹配一次就记为一个访问。当用户查询到来时,查询系统无法利用临时缓存,而该查询对虚拟缓存命中时,则启动缓存替换机制。如果临时缓存中缓存项没有达到上限(考虑到临时缓存集中缓存项的数量应该保持在一个合理的数量,可以设定一个阈值),而虚拟缓存中某项目的访问次数达到一定要求,则可将该项目添加到临时缓存中。如果临时缓存已满,则对临时缓存中所有项目的访问次数与虚拟缓存中该项目的加权访问次数进行排序,其中排序值最小的被淘汰,进入虚拟缓存中。虚拟缓存中项目访问次数加权值范围为(0,1)之间的实数,具体取值根据实际应用而调整。

#### 3.3.1 相关定义

**定义4** 对于临时缓存队列和虚拟缓存队列中的每个缓存项,其访问频率的计算方法定义如下:

$Query\_access\_Frequency = \text{命中次数}(\text{hit\_times}) / \text{最近时间段}(\text{Last\_Timeslice})$

注:根据实际情况,设定  $last\_Timeslice$  的值,这里设定为一个周(7天)

**定义5** 满足进入临时缓存队列的虚拟缓存项  $S_R$  定义为:

对于虚拟缓存集中的缓存项,如果它的访问频率  $Query\_access\_Frequency$  大于或等于设定的阈值  $Access\_Frequency\_Valve$ ,则称该缓存项满足进入临时缓存队列条件。

**定义6** 缓存替换发生的条件定义如下:

存在满足进入临时缓存队列条件的缓存项,而且  $Cache\_num$  等于  $Cache\_num\_Valve$  时,实施替换。

**定义7** 优先等待队列  $Queue\_up$  的定义如下:

在实施替换策略的过程中,由于替换算法的约束,如果  $S_R$  不能进入临时缓存队列,则将移至该队列中,继续进行记频分析。该队列中的缓存项的替换优先权值介于临时缓存和虚拟缓存之间。

#### 3.3.2 缓存项权值的设置

对于临时缓存集和虚拟缓存集的每个缓存项,在替换时机来临时计算出其权值:

$Weight = Cache\_PRI * Query\_access\_Frequency$

注:其中  $Cache\_PRI$  是我们为不同缓存类型设定的优先权值,设定如下:

临时缓存列表中缓存项的  $Cache\_PRI = 1$

虚拟缓存列表中缓存项的  $Cache\_PRI = 0.5$

优先等待队列中缓存项的  $Cache\_PRI = 0.75$

#### 3.3.3 替换算法思路

当缓存替换时机产生时:

1) 对临时缓存集中的各个缓存项,计算出它们各自的权值;

2) 对满足自动添加到临时缓存集中的虚拟缓存项,计算出它们各自的权值;

3) 对优先等待队列中虚拟缓存项,计算出它们各自权值;

4) 根据计算出来的权值对所有临时缓存项和满足自动添加条件的虚拟缓存项及优先等待队列中的虚拟缓存项统一进行排序;缓存项对应的权值越大,表示该缓存项在临时缓存集中存在的意义更大,排序的队列根据权值由大到小排列;

5) 根据  $Cache\_num$  值,检查经过排序的队列的前  $Cache\_num$  位;

6) 如果满足自动添加条件的虚拟缓存项或优先等待队列的缓存项在其中,则将它与排在最后的临时缓存项进行替换;

7) 如果满足自动添加条件的虚拟缓存项不在其中,则将它添加到优先等待队列,继续对它进行记频。

算法描述:

```

If C_replace == true      /* 如果缓存替换发生条件为真时 */
  SR.weight = Cache_PRI * Query_access_Frequency
  /* SR 为满足进入临时缓存队列条件的虚拟缓存项,
     Cache_PRI = 0.5 */
  For each cache item of temporarily queue
    /* 对于临时缓存队列中的每个缓存项 */
    ST.weight = Cache_PRI * Query_access_Frequency
    /* ST 为临时缓存队列中的缓存项, Cache_PRI = 1 */
  For each cache item of pri_wait queue
    /* 对于优先等待队列中的每个缓存项 */
    SU.weight = Cache_PRI * Query_access_Frequency
    /* SU 为优先等待队列中的虚拟缓存项, Cache_PRI = 0.75 */
  For each cache item of dummy and pri_wait queue and SR
    Queue_sort.Sort( ST.weight, SR.weight, SU.weight )
  /* Queue_sort 为根据缓存项的权值由大到小排序后的队列 */
  For each item of Queue_sort
    If QS.number <= Cache_num
      /* QS 为 Queue_sort 队列中的成员, Cache_num 为设定的需要维护
         的缓存规模大小(合理的临时缓存项数目) */
      Queue_maintenance.Push( QS )
      /* Queue_maintenance 为需要维护的缓存项队列 */
    If ( QS.number > Cache_num ) and ( QS ⊆ Queue_temp )
      /* Queue_temp 为临时缓存队列 */
      Queue_replace.Push( QS )
      /* Queue_replace 为需要替换的缓存队列 */
  For each item of Queue_replace
    If QM ⊆ Queue_dummy or QM ⊆ Queue_up
      /* QM 为 Queue_maintenance 队列中的成员, Queue_dummy 为虚
         拟缓存队列 */
      Replace( QM, QR ) /* QR 为 Queue_replace 队列中的成员 */
    Else if QM ⊆ Queue_temp
      num ++
    If num == Cache_num
      Queue_up.Push( SR ) /* Queue_up 为优先等待队列 */

```

#### 3.4 缓存一致性维护

业务系统对数据库服务器的大量写操作使得语义缓存中的数据可能会与数据库中的数据不一致。为了保证语义缓存的有效性,语义缓存需要建立自己的一致性维护机制,确保在更新缓存数据的同时不能改变其对应的语义。语义的引入增

(下转第1864页)

法首先构造跨域过程,通过不断的分解,逐步构造业务过程、业务活动和业务操作,最终将业务操作与业务对象关联起来;自底向上法正好相反,首先依据业务对象的状态转换构造业务操作,然后依据逻辑关系,通过逐步的合并,构造业务活动、业务过程和跨域过程。二者相比,自顶向下法构造的业务模型能够贴近企业管理模式,模型清晰简要,但难以保证模型的完备性和重构性能;自底向上法得到的业务模型相对完备,但构造过程与最终得到的模型都会比较复杂。在业务元素建模完成之后,建模人员将业务元素之间可变的关联关系提取出来,构造上述八类业务规则并采用相应的表达形式进行规则描述。

当依据业务模型构造软件系统时,将这些规则统一的表达为 XML 的形式,运行时的软件系统读取 XML 规则文件并控制系统行为。当企业需求发生变更时,用户可直接修改 XML 规则文件,即可改变系统行为,实现系统重构。限于篇幅,本文将不对该处理过程作详细讨论。

#### 4 结语

传统业务模型强调完备性,而很少考虑重构性。部分研究成果给出了若干面向重构的建模原则,但可操作性较差。另外,大量针对业务规则的研究缺乏与业务模型的结合,难以有效应用于企业建模之中。与以往的研究相比,本文提出的面向重构的业务模型在“持续分解”的原则的指导下,将业务元素的变化转化为元素之间关联关系的变化,从而极大降低

了系统重构时的复杂性。另外,基于该业务模型提取出了八类业务规则,覆盖了模型中各种可能变化的关联关系类型。最后给出的业务模型建模过程也有助于指导建模人员在建模过程中就考虑未来的重构需求,尽可能避免模型设计中的不合理性,使信息系统既能满足企业的业务需求,又易于以后的重构。本文的结果为企业应用系统的分析与设计提供了一种面向重构性能优化和改善的手段,具有重要意义。

#### 参考文献:

- [1] 李群明, 张士廉, 王成恩, 等. 可重构的企业管理信息系统研究[J]. 信息与控制, 2001, 30(7): 630-635.
- [2] 战德臣, 徐晓飞, 李成严. 时间-成本双主线 ERP 管理体系研究[J]. 计算机集成制造系统-CIMS, 2002, 8(8): 635-639.
- [3] ESPRIT Consortium AMICE. CIMOSA: Open System Architecture for CIM[M]. Berlin: Springer-Verlag, 1993.
- [4] DOUMEINGTS G, VALLESPER B, CHEN D. Methodologies for Designing CIM System: a Survey[J]. Computers in Industry, 1995, 25(3): 263-280.
- [5] VAN ES RM, POST HA. Dynamic Enterprise Modeling[M]. Dordrecht: Kluwer, 1996.
- [6] 柴跃廷, 张晓东, 李芳芸. 敏捷信息系统的研究[J]. 计算机集成制造系统-CIMS, 1999, 5(2): 6-10.
- [7] 胡锦敏, 张申生, 余新颖. 基于 ECA 规则和活动分解的工作流模型[J]. 软件学报, 2002, 13(4): 761-767.
- [8] 沈延森. 快速可重构信息系统及其关键技术研究[D]. 南京: 南京航空航天大学, 2001.

表 1 性能测试结果

测试数据规模/GB	查询平均响应时间/s	
	有缓存	无缓存
210	0.162 98	561.462
630	0.193 38	581.665
1050	0.235 17	602.795

(上接第 1845 页)

加了语义缓存的一致性维护的复杂程度。

大规模事务处理系统面临的统计分析业务需求具有一定的规律,因此,我们采用定时刷新的机制来保证语义缓存的一致性。用户在自定义缓存的时候可以指定刷新的周期,当刷新周期到达时,刷新进程就从缓存元数据表中查询缓存项描述信息,重构查询语句,将查询结果写入缓存结果表。

#### 4 性能测试

为了测试语义缓存的性能,选择一个大规模事务处理系统做性能测试<sup>[6]</sup>,对该系统中的原并行查询服务和带语义缓存技术的并行查询服务分别进行测试,对比其处理统计分析类查询的时间,测试语义缓存是否能缩短统计分析的响应时间。并在系统的数据规模分别为 1TB, 630GB 和 210GB 三种情况下进行统计分析查询测试,用以测试数据规模的变化对基于语义缓存的统计分析查询处理时间的影响。其测试环境如下:

查询中间件节点机: 硬件平台 2 × 2G CPU, 4G SDRAM Memory, 3 × 70G Hard Disk; 软件平台为 RedHat7.3 + Oracle 8.1.7 Client + Starbus4.0(基于 CORBA 的事务处理中间件);

数据库节点机(5 台): 硬件平台 2 × 2G CPU, 4G SDRAM Memory, 3 × 70G Hard Disk; 软件平台为 RedHat7.3 + Oracle 8.1.7 Server;

缓存节点机(1 台): 硬件平台 2 × 2G CPU, 4G SDRAM Memory, 3 × 70G Hard Disk; 软件平台为 RedHat7.3 + Oracle 8.1.7 Server;

网络环境: 100M 以太网;

利用例 1 中的统计分析查询进行测试,其测试结果如表 1 所示。

测试结论: 本文提出的中间层语义缓存解决方案对海量数据统计分析查询性能有显著提高,并且数据表明在数据规模增长的情况下,日常的统计分析时间基本不变。

#### 5 结语

本文以海量数据库应用系统为背景,通过对中间层的语义缓存研究,提供了一个适合海量数据统计分析的中间层语义缓存解决方案。其主要是针对海量数据统计分析应用,在将来的工作中,将会逐步考虑更复杂的条件下语义缓存的模型,使其适用于更普遍的海量数据库应用。

#### 参考文献:

- [1] DAR S, FRANKLIN MJ, JONSSON BT. Semantic data caching and replacement[A]. Proceedings of the 22nd VLDB Conference[C]. Mumbai (Bombay), India, 1996. 330-341.
- [2] BASU J. Associative caching in client-server databases[D]. Stanford University, 1998.
- [3] REN Q. Semantic Caching and Query Processing[R]. Technical Report 98CSE-04, Southern Methodist University, 1998.
- [4] Oracle Corporation. Oracle 9i Application Server Database Cache[Z]. 2000.
- [5] LUO Q, KRISHNAMURTHY S, MOHAN C. Middle-tier database caching for e-business[A]. SIGMOD Conference 2002[C]. 2002. 600-611.