移动计算中语义缓存的改进研究

龚玉利[1] 冷文浩[2]
[1](江南大学物联网工程学院 江苏 无锡 214122)
[2](中国船舶科学研究中心 江苏 无锡 214082)

摘 要 针对现有移动计算环境中的语义缓存缺乏位置信息的空间索引,提出一种基于四叉树索引结构的语义缓存及其查询裁剪算法,扩展了传统的语义缓存,实现面向对象的语义缓存。通过模拟实验,对算法的性能进行比较分析。结果表明四叉树语义缓存,降低了平均响应时间、查询比较次数和网络通信负荷。

关键词 移动计算 移动对象数据库 移动对象索引 语义缓存 四叉树

中图分类号 TP399 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2014.02.010

ON IMPROVEMENT OF SEMANTIC CACHE IN MOBILE COMPUTING

Gong Yuli[1] Leng Wenhao[2]
[1](*School of Internet of Things, Jiangnan University, Wuxi 214122, Jiangsu, China*)
[2](*China Ship Scientific Research Center, Wuxi 214082, Jiangsu, China*)

Abstract In existing mobile computing environment, semantic caching lacks the spatial index of location information. In view of this, we propose a quadtree index structure-based semantic caching and its query trimming algorithm, this expands the traditional semantic caching and implements the object-oriented semantic cache. Through simulation experiments, we compare and analyse the performances of the algorithm. Results show that the quadtree semantic cache reduces the average response time, the number of comparisons and the network traffic load.

Keywords Mobile computing Moving object database Moving object index Semantic cache Quadtree

0 引 言

移动计算环境使得人们能在任何时间,任何地点访问信息,另一方面其移动性,低带宽,频繁断线和移动设备有限的资源,成为移动计算应用发展的瓶颈。为了解决这些问题,方法之一是利用语义缓存技术。

语义缓存是一种基于结果集及其描述的缓存技术,其内容包括查询结果和语义查询描述。移动设备或客户端能够判别是否可以在本地回答请求查询或是通过推断缓存里的语义来决定访问服务器获得查询结果。因此,通过语义缓存技术,移动计算应用减少访问数据库服务器的时间,因而减少网络流量,提高响应时间。

移动计算系统是一种分布式系统。分布式应用程序往往涉及不同的异构资源。因而为实现分布式系统,需要提出不同的方法处理[1,2]。随着手机和通信技术的快速发展,移动计算设备越来越多的数据或资料涉及到的位置。在现实生活中,有很多位置相关的数据。空间索引结构可以在基于位置服务管理的语义缓存中发挥作用,例如移动地图服务。

为此,我们提出了一种基于四叉树的管理模型来组织语义缓存,讨论查询语义,实现语义缓存查询算法,并提供一些提高查询处理过程性能实验结果。

1 相关工作

以往的研究中,语义缓存专注于查询处理,更换合并策略的研究。文献[3]研究查询处理和语义缓存替换策略在移动计算位置相关数据的管理。文献[4]中研究了利用新型的硬件和数据库服务器对语义缓存再访性能的影响。文献[5]中提出了基于语义缓存的移动客户端缓存设计,提出缓存项合并规则,并讨论在不同情况下的处理规则。文献[6]提出了语义缓存的正式定义和详细的查询处理策略。文献[7]设计了一个详尽的语义缓存架构能够响应所有类型的查询,其包含缓存管理算法和缓存组单元替换算法。文献[8]提出了支持聚合查询和两种聚合查询匹配算法的语义缓存模型。文献[9]提出了一种基于谓词分类的语义缓存查询调整算法。

目前,移动计算中语义缓存的研究重点在维护缓存一致性方法,对中间件中语义缓存的研究和分布式的语义缓存的研究。在目前的研究中,面向对象的语义缓存的实施尚未讨论,所以,面向对象的语义缓存的实现,使语义缓存更容易应用到不同的计算应用中,如分布式计算模型,生物启发计算[10]等等。

我们的工作在以往研究的基础上,提出了一种基于四叉树

收稿日期:2012-08-26。国家自然科学基金—联合资助基金项目(U0970125)。**龚玉利**,硕士生,主研领域:移动 GIS。**冷文浩**,研究员。

的隐式索引结构管理语义缓存的位置属性,支持查询处理的匹配和调整算法,讨论一种面向对象的语义缓存的实现方法。

2 语义缓存的管理模式

2.1 语义缓存的内容和组织

语义缓存不仅存储查询的实际信息,还存储查询描述。在本文中,我们用到文献[4,5]中一些语义缓存概念,并将其进一步扩展。

定义 1 查询 Q 定义为 Q = <Qr,Qa,Qp,Qt>,Qr 定义为查询对应的关系的集合,Qa 定义为在查询中出现的属性集合,Qp 是查询中条件谓词,Qt 定义为查询提交的时间。R,A,P 的意思与文献[5]中定义的一致。查询谓词 Qp,表示为三段式{attribute,operation,Data},其中 attribute 为关系中对应属性;operation 取值{=,<,>,≤,≥,like,not like};data 取值{string,integer,float,double,Boolean}。

定义 2 语义缓存 SRC,定义为 SRC = <Sq,Sl,Sc>,这里 Sq 定义为缓存的语义,表示为<Qr,Qa,Qp,Qt>,Sl 代表位置属性,Sc 定义为语义缓存片段的集合。

定义 3 SRC 部分扩展,定义为 SRC' = <S'q,Sl,Sc>,S'q = <Qr,Q'a,Qp,Qt>,Q'a = Qa ∪ Qp (Attribute),Qp (Attribute)表示出现在谓词 Qp 中的属性集合。

定义 4 SRC 完全扩展,定义为 SRC^* = <S^*q,Sl,Sc>,S^*q = <Qr,Q^*a,Qp,Qt>,Q^*a 是关系属性 Qr 中的所有属性。

移动计算最重要的特征是移动终端的移动性。许多学者提出不同的方法管理和使用大量的空间和位置信息。在使用基于位置的服务时,位置属性是影响缓存数据的重要属性。我们提出使用索引技术,管理语义缓存在移动计算环境,尤其是在移动地图服务系统中的应用,拓展了传统四叉树能在移动计算中处理移动物体。在线性四叉树中,不能再分的结点称为叶子节点,可再分的结点称为树枝节点。一个结点最多有 4 个子节点,每个子结点按照区域分解可命名为 *SiWi*、*SiEi*、*NiWi* 或 *NiEi*,分别表示父结点所在区域的西南象限、东南象限、西北象限、东北象限,其对应的对象分别称为 *SiWi* 像元、*SiEi* 像元、*NiWi* 像元和 *NiEi* 像元,其中,$i \in \{1,2,\cdots\}$ 表示线性四叉树的层高或深度。

如图 1 所示,根据四叉树法二维空间每次递归地分为四个部分,即东北,西北,东南和西南四个区域,直到在每个部分的对象的数量不大于阈值。在本文中,一个对象对应一段语义段,多个对象对应的多个语义段。四叉树是一种简单且不平衡的树,高密度区域子树的深度是比密度低的区域树深度要大。因而在空间搜索时它能提高性能。

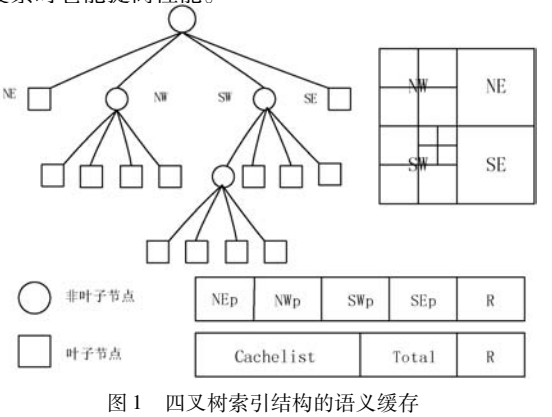

图 1 四叉树索引结构的语义缓存

我们利用四叉树来索引语义缓存。非叶节点由 Not_LeafNode(NEp,NWp,SWp,SEp, R)表示,NEp,NWp,SWp,SEp 是分别指向东北,西北,西南和东南的指针。R 表示位置属性的协调范围,由(XLT,YLT,XBR,YBR)表示,<XLT,YLT>左上方协调的范围和<XBR,YBR>右下方协调的范围内。语义缓存下四叉树结构的叶节点表示为 LeafNode(CacheList,Total,R),其中 CacheList 是语义缓存链表结构的头指针,Total 表示语义缓存数量的最大值,R 与非叶节点中 R 的值是一样的。通过四叉树,位置属性语义缓存段能够被索引,这一索引作为一种位置属性的区域索引。这种管理模式被称为基于四叉树语义缓存(简称 QSC)模型。

对于缓存的空间数据组织来说,关键问题是空间数据的索引与检索,空间索引的性能优劣直接影响语义缓存整体性能,由于四叉树结构的生成和维护相对比较简单,因而对于缓存的空间数据对象分布比较均匀时,基于四叉树索引结构的语义缓存可以获得比较高的空间数据插入和查询效率。

Cachelist 是一个列表结构,管理缓存。每个缓存都包含地图指定区域上的大量的记录或数据。缓存是部分或完全扩展的语义缓存中的一种,它包括提出的查询和相应的结果。部分扩展和充分扩展的不同在于,前者继承了出现在 SQL 语句 where 条件中的属性,而后者则包括可以通过访问数据库中数据字典所获取的所有属性的关系。

2.2 查询匹配和裁剪算法

查询处理的主要思想如下,缓存每个查询被分成两个不相交部分:一种可以完全使用缓存中存在数据来响应,而另一种需要从服务器获得返回值来响应。这两种方式分别称为探寻查询和剩余查询。探寻查询的查询结果已经在高速缓存中,并可以立即返回给用户。而剩余查询需要从服务器中取出,并进入缓存,然后才可以返回给用户。

通常情况下,语义缓存有许多缓存段。如果使用链表结构来管理缓存段,处理过程可以大致描绘如下:在前一段缓存段查询裁剪后,下一个缓存段的查询裁剪开始。此过程循环执行,直到没有缓存段响应查询。而最终那些剩余查询将被发送到服务器进行处理。我们在本文中称此方法为列表法。如算法 1 所给出列表法的实施。

```
Algorithm 1  List-Method (Query Q, Cache * CacheList, Dataset DS)
/* o generate the results DS for the query Q from the semantic cache. */
Input: Q: the query;
CacheList: the pointer to the head of the list structure for the semantic cache.
Output: DS: the set of query result from the semantic cache.
Procedure
Step 1 var Query probeQuery = Q;
Query remainderQuery = null;
var DataSet dsi = null; DataSet DS = null;
Step 2 For (each cache in CacheList) Do
{If (isIntersect (cache.predicate,
probeQuery.predicate) == ture)
then jump to Step 3;
Else jump to Step 2;}
Step 3 calculating the probeQuery and remainderQuery;
Step 4 For (each data in cache.dataset) Do
{If (compare(probeQuery.predicate, data)! =0)
```

```
then ProbeDS. add( data) ;
//Constructing the ProbeDS from cache. dataset //through comparing between //probeQuery. predicate and cache. dataset.
}
Step 5 If( remainderQuery! = null)
Then probeQuery = remainderQuery,
jump to Step 2;
Else jump to Step 6;
Step 6 If( remainderQuery! = null)
Then remainderDS =
getDatafromServer ( remianderQuery) ;
/ * necessary, send the remainder query to erver and get data from server * /
Step7DS. add( probeDS) ;
DS. add( remainderDS) ; return.
```

链表法管理缓存是充分利用链表的数据结构,对于链表的缓存分段循环检测,直到整条链表中在缓存中没有响应为止,其特点是数据结构简单,但是在其过程中存在很多对服务器的无效访问和不必要的查询和比较时间。

本文中,我们提出了一种新的方法称为四叉树法语义缓存管理。其主要思想如下:这个过程开始于整个树的根。比较当前查询中和节点的 R 值的交集,循环执行,直到比较的节点是叶节点或当前查询中和节点的 R 值有没有交集。如果该节点是叶节点,列表的处理程序和上面描述的链表结构的算法是一样的。没有交集的意思是指没有可以帮助回应查询的缓存段。否则,查询可以回答语义缓存的部分或全部。如算法 2 所给出四叉树法语义缓存管理实施。

```
Algorithm 2　Quadtree-Method ( Query Q, node * quadtree, DataSet DS)
/ * to generate the results DS for the query Q from the semantic cache which is indexed by the quadtree. * /
Input: Q: the query; quadtree: the pointer to the root of the quadtree structure for the semantic cache
Output: DS: the set of query result from the semantic cache
Procedure
Step 1 var node * currentNode = quadtree;
var Query probeQuery = Q;
Query remainderQuery = null;
Step 2 If( isIntersect( Q. SL, currentNode. R) == true)
Then jump to Step 3;
Step 3 If( currentNode == Not_LeafNode)
Then
Quadtree-Method( Q, currentNode- > NEp, DS) ;
Quadtree-Method( Q, currentNode- > NWP, DS) ;
Quadtree-Method( Q, currentNode- > SEp, DS) ;
Quadtree-Method( Q, currentNode- > SWp, DS) ;
Else jump to Step 4;
Step 4 For( each cache in currentNode. CacheList) Do
If( isIntersect( Q. SL, cache. SL)
Then DS. add( getData( Q, cache) ) ;
//get the probe query data and add to the DS;
Step 5 Calculating the remainderQuery;
Step 6 If( remainderQuery! = null)
Then remainderDS =
getDatafromServer( remianderQuery) ;
/ * If necessary, send the remainder query
to server and get data from server * /
Step 7 DS. add( remainderDS) ; return.
```

相比列表法,四叉树法更有效,因为它不进行查询和每个缓存段之间的比较,并且只查询到四叉树的叶节点时才计算探寻查询和剩余查询,因此,它减少了很多不必要的查询,同时也降低了响应时间。另一方面查询剪裁越复杂,探寻查询和剩余查询也会越复杂。使用四叉树法不仅加快了搜索,还大幅降低查询剪裁的时间,查询和缓存之间的比较,从而提高查询在移动计算环境的效率。

3　语义缓存的实现管理

语义缓存的实施结构包括 5 类,语义缓存 Cache 中定义缓存的语义和缓存片段集合,在缓存语义 Query 中,定义了查询对应的关系的集合,其中 Attribute 为在查询中出现的属性集合,Predicate 是查询中条件谓词,如图 2 所示。

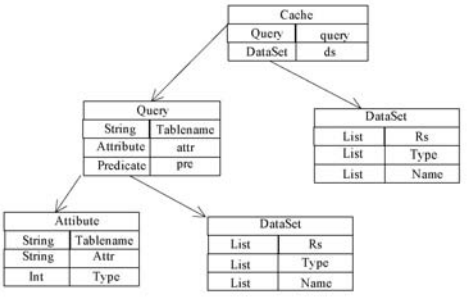

图 2　在执行语义缓存的类

对于客户发出的任何一条 select-from-Where 查询语句,如果 Where 子句是通过逻辑运算符 and 连接的任意多个由属性和常量组成的比较表达式,我们称该语句为可缓存查询,定义为 SRC = < Sq, Sl, Sc > ,这里 Sq 定义为缓存的语义,表示的是查询 < Qr, Qa, Qp, Qt > ,Sl 代表位置属性,Sc 定义为语义缓存片段的集合。之所以只缓存符合上述要求的查询,主要出于性能和代价方面的考虑。select-from-Where 结构的查询语句是最常用的查询语句,对于有更复杂的查询语句的查询,其结果再次利用可能性小,缓存下来的意义不大。对于合理的语义缓存语句再由缓存的扩展粒度,采用部分或者全部扩展将查询中出现的属性集合进行扩展,为了简化对象的管理,可以采用全部扩展。

语义缓存的内容通过一组缓存项描述,对于每次新加入缓存的数据,可以按其查询语句生成缓存描述项与之对应,每当有新的数据进入,或有数据被替换,缓存的内容及缓存项描述都要做相应的改变,每次客户的查询要从缓存中获取结果都先要进行匹配。选择本文提出来的四叉树来索引语义缓存,执行查询剪裁。

具体的语义缓存实现管理原理通过下面实例来说明。假设有查询 Q1{ Select a, b from Table 1 where a > = 15 and d < 200}。我们采用语义缓存完全扩展方法,这样在查询中出现的属性扩展为所有的属性,集合客户端通过访问数据字典取得表 1 的四个属性 a,b,c,d。

表 1　示例关系

a:int	b:string	c:long	d:double
13	35	67	72
18	39	101	160
21	58	136	218
…	…	…	…

将 Q1 拓展为 Q′1{ Select a, b, c, d from Table 1 where a > = 15 and d < 200},然后客户端发送新的查询 Q′1 到服务器,

并且得到服务器的结果。这一实例过程如图 3 所示。

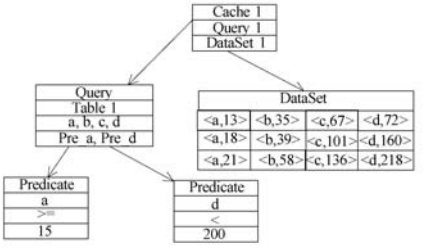

图 3 查询 Q′1 的实例

一旦缓存通过之前查询的内容或者语义缓存段而建立起来,下一查询可以通过使用缓存来执行。如果不使用索引缓存,查询的匹配和剪裁算法可以采用链表方法(算法 1)。当语义缓存中的位置是要明确确定下来的时候,我们可以使用索引结构来管理 LDD 的位置属性。在这种情况下,四叉树算法(算法 2)可以在建立语义缓存的四叉树索引后使用。由于采用语义缓存完全查询扩展,类属性对象不需要创建,因而简化了对象的管理。

4 实验与分析

4.1 实验环境

该仿真系统由服务器,移动客户端和无线网络组成,架构如图 4 所示。当移动客户端提交查询请求时,它会首先在移动客户端处理。如果语义缓存包含所有所需的数据,则查询过程在本地完成。如果缓存只是满足部分查询,则将剩余查询通过网络提交到服务器处理。J2ME 应用在语义缓存算法基础上实现的。

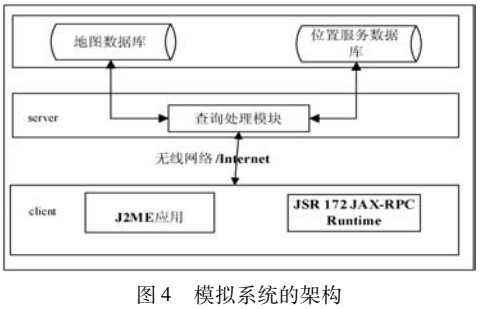

图 4 模拟系统的架构

在模拟的移动计算环境中服务器端主要对请求处理,根据客户端请求查询数据库服务器,其中更新生成器随机产生更新操作,模拟对服务器端的数据进行更新。客户端查询处理模块进行查询裁剪,并将剩余查询语义及查询结果分别写入相应的数据表中;查询生成器随机产生用户查询,模拟客户端的查询请求。

表 2 显示了服务器端和移动客户端的主要实验环境参数。

表 2 实验环境参数

参数	值或类型	描述
CPU	Dual-Core 2.6GHz	处理频率
内存	4G	内存大小
数据库	MySql	数据库
数据数量	10 000	数据库的数据量
缓存数量	0	初始缓存数量
查询次数	10	各组查询的次数
带宽	10 M/25 kb	网络带宽
客户端	MOTO me727	移动客户端

实验测试数据是由无锡市中心城区的地图生成的数据,其中包含写字楼,购物中心,住宅区、生活设施区域等。在本文中,测试用例是一组查询,其中每个查询 Q 从当前位置 L 开始在一定范围 R 内查询类型为 O 的目标对象信息。查询和相应的结果使用上文中提出的完全扩展的缓存。

L 当前提交查询的移动客户端的位置。

R 从 0 到 1 000 的随机取值

O 是被查询对象的类型

例如,L = 火车站,R = 500,O = 餐馆,则 Q 为以火车站为中心 500 米范围内的所有餐馆。

在测试中我们选择几个性能指标进行实验对比。

1)响应时间。这是一个在查询处理过程中的主要评价因素。

2)查询比较次数。在查询处理过程中,在查询语义和语义缓存数据之间有多次比较。因此,查询比较的次数也是一个重要的评价指标。

3)服务器传输的数据量。带宽是移动计算环境面临的最大挑战之一,在移动计算环境中,从服务器传输的数据量在移动计算查询处理性能中起到重要的作用。

4)服务器的连接次数。针对移动计算高频率断线环境,较少的连接次数意味着更好的环境适应能力。

4.2 实验结果分析

实验比较以上四个性能指标(第 4.1 节中所述)在无缓存的索引模型,传统的语义缓存模型(列表管理模式)和四叉树的语义缓中的评价结果。在图 5 至图 8,横轴表示查询的次数,纵轴分别表示响应时间,查询比较次数,从服务器传输的数据量和连接服务器次数。测试结果是平均执行时间。

在图 5 中,比较无缓存模型,传统的语义缓存模型和四叉树的语义缓存模型的响应时间。一般来说,数据量较大的查询和处理过程,需要更多的时间。从图 5 中,我们可以看出,无缓存模式响应时间的斜率大于语义缓存模型,四叉树的语义缓存模型的响应时间比传统的语义缓存模型少。这是因为无论是查询时间还是从服务器返回的数据量,在四叉树的语义缓存模型中都比传统的语义缓存模型少。

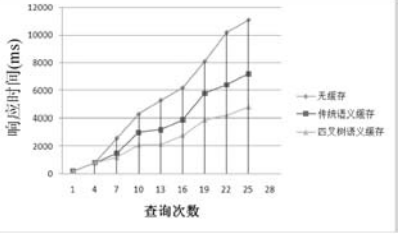

图 5 响应时间比较

在图 6 中,无缓存模型查询比较次数与查询次数成正比,而缓存模型曲线的斜率很小。更重要的是,四叉树的语义缓存模型的查询比较次数小于传统语义缓存模型。这是因为位置依赖查询有很大的语义相似性,因此使用四叉树语义缓存模型可以消除不必要的比较。实验结果表明,四叉树的语义缓存模型,可以更好地利用缓存,提高查询效率。

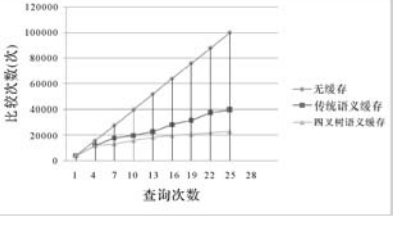

图 6 查询比较次数的比较

(下转第 115 页)

4 结　语

家具行业开料排样云服务系统将优化排样算法、虚拟化技术、云服务封装等技术进行集成,以云形式对外提供材料开料排样优化计算服务,特别适合于家具产业集群度高的地区。另外,云服务平台可进一步联合中小型家具企业,逐步建立起家具开料设备的数据中心,实现以开料设备为主的制造资源的云端共享,探索家具开料环节的云端制造模式。

参　考　文　献

[1] Lodi A, Martello S, Monaci M. Two-dimensional Packingproblems: A survey[J]. European Journal of Operational Research,2002,141(2): 241 -252.

[2] Leung J,Tam T,Wong C S, et al. Packing Squaresinto Square[J]. Journal of Parallel and Distributed Computing, 1990, 10 (3): 271 -275.

[3] Hifi M, Saadi T. A parallel algorithm for two-staged two-dimensional fixed-orientation cutting problems[J]. Computational Optimization and Applications,2012,51(2):783 -807.

[4] Hifi M, Hallah R. Approximate algorithms for constrained circular cutting problems[J]. Computers & Operations Research,2004,31(5): 675 -694.

[5] 洪灵,王耘. 一种不规则零件排样的快速解码算法[J]. 计算机辅助设计与图形学学报,2005,17(11):2465 -2470.

[6] Stephen C L,Zhang Defu, Kwang M S. A two-stage intelligent search algorithm for the two-dimensional strip packing problem[J]. European Journal of Operational Research,2011,215:57 -69.

[7] Leung S C H,Zhang D. A fast layer-based heuristic for non-guillotine strip packing[J]. Expert System with Application, 2011, 38: 13032 -13042.

[8] 何琨, 黄文奇. 三维矩形 Packing 问题的拟人求解算法[J]. 中国科学,2010,40(12):1586 -1591.

[9] 陈康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009, 20(5):1337 -1348.

[10] 刘鹏. 云计算[M]. 3 版. 北京:电子工业出版社,2011.

(上接第 40 页)

在图 7 中,横轴是查询次数,纵轴是无缓存模型和语义缓存模型从服务器传输的数据量。由于传统语义缓存模型和四叉树语义缓存模型有相同的实验特点,所以他们都被称为语义缓存。如果没有提及,语义缓存模型即代表两者。从图 7 中我们可以看到,在没有缓存模型,在网络上传输的数据量非常大,几乎与查询次数成正比。在语义缓存模型中传输的数据相同的情况下,斜率显著降低。这是因为所需的查询结果全部或部分包含在语义缓存中。实验结果表明,查询处理的语义缓存模型,可以充分利用缓存资源,减少网络负载。

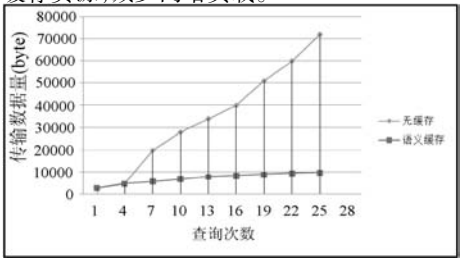

图 7　服务器传输的数据量比较

在图 8 中,该服务器连接次数是与没有缓存模型的查询的次数成正比,因为每次查询必须访问到服务器。而语义缓存模型连接次数较少,这是因为所需的查询结果全部或部分包含在语义缓存中。实验结果表明,语义缓存模型能很好地支持查询时经常断线发生。

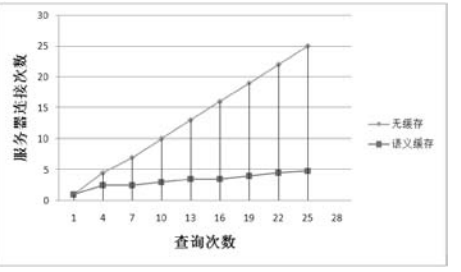

图 8　服务器连接次数比较

5 结　语

移动计算环境低带宽、网络断接频繁、设备移动性和设备资源有限给移动客户端带来很多挑战。在本文中,我们提出了一种基于四叉树的语义缓存模型,设计了查询匹配算法以及其面向对象的实现。实验证明,我们提出来的模型获得良好的性能并且降低了网络的通信量,减少了连接到服务器的次数和平均响应时间。今后的工作是把重点放在以此为基础的分布式查询优化。

参　考　文　献

[1] Frei R, Barata J. Distributed systems-from natural to engineered: three phases of inspiration by nature[J]. International Journal of Bio-Inspired Computation,2010(2):258 -270.

[2] Pitoura E, Chrysanthis P K. Caching and replication in mobile data management[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,2007,30(3):13 -20.

[3] Ren Q, Dunham M H. Using semantic caching to manage location dependent data in mobile computing[C]//Proceedings of MOBICOM 2000, 2000:210 -221.

[4] Jónsson B T, Arinbjarnar M, Tórsson B. Performance and overhead of semantic cache management[J]. ACM Trans Internet Techn, 2006,6 (3):302 -331.

[5] 吴婷婷, 苏武运,等. 移动查询缓存处理的研究[J]. 计算机研究与发展,2004,41(1):187 -193.

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

[7] Safaeei A A, Haghjoo M, Abdi S. Semantic cache schema for query processing in mobile databases[C]//Proceedings of ICDIM 2008, 2008:644 -649.

[8] 蔡建宇,吴泉源,等. 语义缓存的聚集查询匹配研究[J]. 计算机研究与发展,2006,43(12):2124 -2130.

[9] 李东,杨小鹏,罗鹏飞. 基于谓词分类的语义缓存查询裁剪[J]. 华南理工大学学报:自然科学版,2008,36(1):44 -49.

[10] Akerkar R,Sajja PS. Bio-inspired computing:constituents and challenges[J]. International Journal of Bio-Inspired Computation, 2009, 10 (3):135 -150.