面向 SaaS 应用基于键值对模式的多租户索引研究

孔兰菊 李庆忠 史玉良 王 学
(山东大学计算机学院 济南 250101)

摘 要 面向 SaaS 应用的多租户数据库为满足租户的数据隔离和按需定制的需求,需要提供支持隔离和易于定制的数据存储机制及索引机制.基于键值对存储方式,提出元数据驱动的映射表索引模型,该模型根据租户定制需求,为租户业务数据形成各自的索引元数据,通过元数据驱动实现了索引数据的隔离及定制效果;给出索引的维护策略,根据租户数据访问请求进行索引切片,以逐渐细化的索引切片作为数据访问的基本单位,快速返回租户结果集.实验结果表明,该方案在数据访问分布均衡的情况下,使索引维护及数据访问具有较好的总体性能.

关键词 多租户;索引;键值对;SaaS;结构化数据
中图法分类号 TP392 **DOI 号**: 10.3724/SP.J.1016.2010.02239

Research on Index of Multi-Tenant Based on Key-Values for SaaS Application

KONG Lan-Ju LI Qing-Zhong SHI Yu-Liang WANG Xue
(*School of Computer Science and Technology, Shandong University, Jinan* 250101)

Abstract In order to excellently support SaaS application, multi-tenant database system needs to meet the tenants requirement of isolation and on-demand customization, and then to provide data storage mechanism and index mechanism that supporting isolation and flexibility. Based on key-values model, this paper proposes a meta-data driven indexing mechanism, according to tenant customization requirements, the model constructs respective index metadata for business data of the tenants, and then achieves isolation & customization effects through metadata-driven mechanism; while the index maintenance strategies are given, in response to tenants access requests, the model forms and slices the index data as the basic unit and return quickly the result sets. In this paper, detailed experimental results show that index maintenance and data access of this program work with good performance under normal conditions.

Keywords multi-tenant; index; key-values; SaaS; structured data

1 引 言

在面向 SaaS[1-2] 的应用中,多租户数据库需要提供租户之间的数据隔离及按需定制功能,同时,还要在性能上让用户有比较好的体验.这对数据的存储和索引机制提出了挑战.多租户数据库突破了传统的关系数据库、对象数据库的定义,也不同于 Hbase 这样面向非结构化数据搜索工作的分布式数据库系统.

多租户数据库存储[1-3]已经成为热点,但是索引方面[4-8]的研究还比较少,目前只有 salesforce 给出

收稿日期:2010-06-11.本课题得到国家自然科学基金(90818001,61003253)、国家科技支撑计划(2009BAH44B02,2009BAH44B04)、山东省自然科学基金(ZR2010FQ026、2009ZRB019YT,2009ZRB019RW)、山东省科技攻关计划(2010GGX10105,2009GG10001002)和山东大学自主创新基金(2009TS030)资助.**孔兰菊**,女,1978 年生,博士研究生,讲师,主要研究方向包括数据库、半结构化数据管理. E-mail: klj@sdu.edu.cn. **李庆忠**,男,1965 年生,博士,教授,博士生导师,主要研究领域为大规模网络数据管理及 Web 数据集成. **史玉良**,男,1978 年生,博士,讲师,主要研究方向为服务计算及可信计算. **王 学**,男,1987 年生,硕士研究生,主要研究方向为数据库.

了稀疏表模式下的数据透视表索引以及云数据管理基于 HBase 存储的多维索引等. 稀疏表方式在支持租户定制方面受 DBMS 的限制,而 HBase 存储主要面向非结构化数据处理. 本文的研究背景为 CRM、HR 等系统,以结构化数据为主. 综合国内外研究成果,本文采取了键值对模式的存储方式,该存储方式能完美地支持租户定制及数据隔离,并且非常适合于描述结构化数据,面对成千上万的租户,如何实现索引的定制及索引数据的隔离存储都成了需要解决的问题.

在现阶段,面向 SaaS 应用的多租户数据管理已经成为国内外研究的热点. 文献[1-2]针对数据安全性、数据可扩展性等方面提出并分析了一些设计模式;文献[3]提出了 SaaS 应用的体系架构,然后提出了基于 SaaS 模式的企业公共服务平台的 3 个紧密相连的模型——多用户数据模型、元数据管理模型和安全服务模型;文献[4]探讨了数据仓库应用在云数据中改进索引模式以减少无效处理时间,提高命中率;文献[5]探讨了多租户数据层如何支持多租户服务计算;文献[6]提出了折叠表方式,并对各种数据管理方式得出了较详细的性能数据文献;文献[7]采用了预定义字段机制支持扩展,在查询方面提出了元数据驱动、外部搜索引擎、次优搜索机制等理念;文献[8]提出了如何使用 XML 来支持数据定制及驱动 ECA 模型;文献[9]更是在 2009 年的 SIGMOD 上讨论了目前主要的集中多租户数据管理方式,包括稀疏表、扩展表、XML、键值对等. 该文献经过全面地论证和分析后认为,理想的多租户数据库尚未出现,还有很多的问题需要研究,本文主要探讨了索引方面的关键技术.

传统的索引方式,在多租户数据库模式下,可以按照 objectID、columnNAME 及 value 建立复合索引,针对 key_search 这种模式,可以迅速定位到某些属性,并且重组成一个元组;对于 key_search 的范围查询,通过 key 定位到某个元组,它需要大量的连接才能反回结果集,存在性能问题;对于复合索引里边的次关键字的查询,如果想快速查询,就必须再次创建复合索引,但存储空间巨大,否则查询性能严重不足.

Column_store[10-13]是面向数据仓库的,基本上是以读为主的应用. 数据仓库有着特殊的背景,它由维度和事实构成,维度通常数量有限,维度的组合虽然多,但是最终形成的方体为稀疏型的. 在这种前提下,monetDB 数据库中提出了 cracking[14-15]的策略,在查询的时候,建立查询列的副本,并按照 avl 树的思想基于每次查询动态调整元组顺序,这是查询驱动的思想. 对于以事务处理为主的多租户应用,每次更新都要及时地反应出来,这种查询驱动的方式存在着明显的缺陷. 本文更多地关注了数据库更新时的索引维护策略.

Hbase[16]是一个类似 Bigtable 的分布式数据库,大部分特性和 Bigtable 一样,是一个稀疏的、长期存储的、多维度的、排序的映射表. 这张表的索引是行关键字、列关键字和时间戳. 每个值是一个不解释的字符数组,数据都是字符串,不支持数据类型. 同一个关系里面的每一行数据都可以有截然不同的属性,为动态定制属性提供了一种手段. Hbase 存在的问题有两点,首先 Hbase 不是多租户的,没有提供隔离机制;其次,Hbase 面向的是非结构化数据,每个值是一个不解释的字符数组,擅长于关键字搜索,它不可能提供基于某个属性组的索引,而结构化数据事务处理需要确切的解释每个属性的含义并对其进行维护. 本文更多地讨论了结构化数据的索引机制.

稀疏表方式建立了数据透视表作为索引,比较有代表性的 Salesforce 就采取了这种方式. 这种索引方式解决了索引列数据对象同质的问题,但是他并没有深入探讨透视表与稀疏表的数据同步问题,也无法解决透视表数量暴涨的问题.

本文基于键值对存储模式,分析了现有索引的局限性,结合稀疏表模式下数据透视表的思想,引入了数据映射表索引机制及切片策略,提出了一种用于多租户数据库的面向结构化数据的映射表索引结构及索引维护策略. 该索引结构能有效地支持多租户定制及隔离特征,并能提供较好的访问性能.

本文第 2 节给出键值对传统索引机制在 SaaS 应用的问题分析,介绍基于键值对模式的多租户映射表索引模型;第 3 节讨论索引更新策略;第 4 节给出上述算法的实验验证;第 5 节给出相关工作;第 6 节给出本文的结论与展望.

2 基于键值对模式的多租户索引模型

2.1 键值对模式介绍

在 SaaS 模式下,数据库不但要提供极高的性能,还要支持租户隔离而且富于扩展性. 多租户数据库要根据环境的不同,比如不同的租户、不同的工作流程、不同的服务水平协议等提供易变性. 结合国内

外研究现状,本文选择了键值对存储方式来存储业务数据.

键值对方式能提供较好的易变性,在键值对方式下,每个属性都存储在一个长而窄的关系里(如图 1).租户根据自己业务的需要,自由定制所需数据项,租户的定制信息被保存在元数据(Metadata)里.租户定制的数据项经过逻辑层与存储层模式映射后,被统一保存在数据关系里(Data Table).对应于租户数据可以建立传统的索引(Index),如 B^+ 树,以方便数据存取.

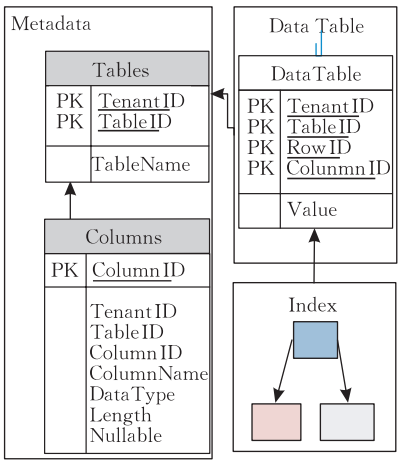

图 1　基于键值对模式的多租户数据库

如表 1 所示,租户从逻辑层看到的仍然是传统关系模式的表格,但是在实际的键值对存储模式里,基于元数据信息,该关系通过模式映射机制被透明地转换到 DataTable(表 2)中.

表 1　租户 001 从逻辑层看到的关系 *R*

RowID	*A*	*B*	*C*	*D*
1	5	2	张三	2008
2	1	3	李四	2009
3	8	9	王五	2004
4	4	1	赵六	2003

表 2　存储层的关系 DataTable

TenantID	*TableID*	*RowID*	*ColumnID*	*Value*
001	*R*	1	*A*	5
001	*R*	1	*B*	2
001	*R*	1	*C*	张三
001	*R*	1	*D*	2008
001	*R*	2	*A*	1
001	*R*	2	*B*	3
…	…	…	…	…
002	*R*	1	*B*	15
…	…	…	…	…

在键值对存储下,可以为业务数据建立全局索引,如(*TenantID*, *TableID*, *ColumnID*, *Value*).当存取单个属性时,根据索引能直接进行操作.当用户试图根据属性 a 访问 b 时,数据库中就没有合适的索引可以使用,只能进行顺序扫描.在键值对存储模式下,即便建立多租户模式下常用的数据透视表索引,也必须要进行多个分支的查找,才能同时存取属性 a 和 b.

在键值对存储下,也可以为每个租户建立数据分区以支持数据隔离,同时按分区建立局部索引实现索引数据隔离.分区机制能有效地支持隔离,但并没有提供对多租户特征的支持,每增加一个租户都要显式地执行分区表定义及索引定义,受数据库模式动态改变及分区表数目上限的限制,无法满足多租户数据管理的需求.

根据以上分析可以看出,直接把传统索引应用于键值对模式的多租户数据库面临着许多问题,概括起来主要有以下几个方面:

(1)传统索引是在关系的某个属性组上建立的索引,该索引键值所包含的数据对象应该是同一类型的数据,这不符合基于元数据的键值对存储模式的特点.

键值对存储模式利用元数据存储租户的数据模式信息,而所有的业务数据都被存在一个纵向生长的关系里,它的数据类型是通用的数据类型,互相之间实质上拥有不同的数据类型,不具有可比性.同时,由于关系里只有一个通用属性,也将无法自然地支持多个属性的联合索引.

(2)现存索引缺乏对多租户标识的直接支持,而此标识是数据对象是否显示的决定因子.

虽然可以通过把多租户标识直接包含到索引的键值里去,但这种大重复量的标识信息势必会增大索引的空间需求,并且也无法自然地支持租户之间索引隔离存储的需求.

综上所述,目前索引只能用于多租户数据库中按键值存取某个元组的情况,无法直接支持租户隔离、租户模式不同、租户按照范围进行存取的情况.本文提出了键值对模式下的多租户索引模型.

2.2　基于键值对模式的多租户索引模型

本文对键值对模式的多租户数据库进行了扩展,增加了基于元数据的映射表索引机制(如图 2 虚线框内),索引模型包含索引元数据和索引数据,索引元数据保存在 *MappingColumn* 及 *MappingSet* 中,索引数据保存在 *Mapping Index* 及 *Mapping-Pending* 中.索引元数据根据定制可纵向生长,不会受限于租户属性的数目;索引数据由元数据驱动,同

样可纵向生长,不会受限于租户业务数据量的影响.本文映射表索引存储于系统缓存(Cache)中,并没有直接实现于关系数据库的底层.

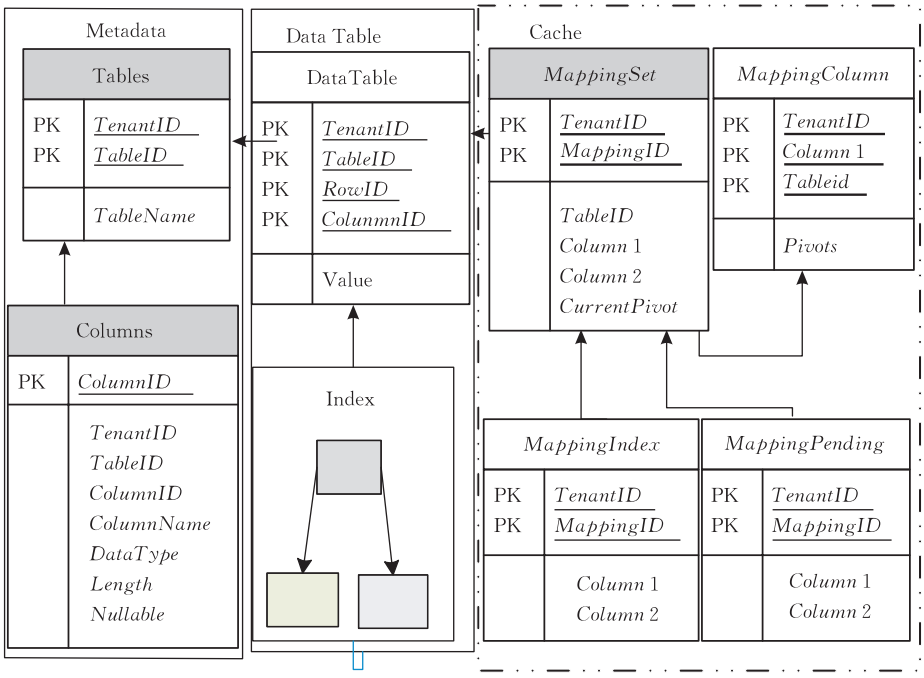

图 2 扩展后的多租户数据库模型

下边给出本模型相关的定义:

定义 1. *Metadata*.

表示元数据,用于保存各租户的数据模式,本模型中用关系表 *Tables*、*Columns*、*MappingColumn*、*MappingSet* 存储.元数据为模式映射提供支持,本文基于元数据把用户请求透明地转换为键值对模式下的操作.

Tables 存储关系定义,*Columns* 存储关系的属性定义,*MappingColumn* 存储所有的索引列及其渐进式的索引更新过程,*MappingSet* 存储与索引列相关的映射定义.本方案中以顺序存储的切片点列表(Pivots)来表示索引更新过程,用 *CurrentPivot* 表示某个映射当前的切片点,在本模型中一般是数据访问请求的参数.

$$Metadata=\{Tables,Columns,MappingColumn,MappingSet\},$$

$$Tables=(TenantID,TableID,Tablename),$$

$$Columns=(TenantID,TableID,ColumnID,ColumnName,DataType,Length,Nullable),$$

$$MappingColumn=(TenantID,TableID,Column1,Pivots),$$

$$MappingSet=(TenantID,MappingID,TableID,Column1,Column2,CurrentPivot),$$

$$Pivots=\{Value\mid Value\in\Pi_{Column1}(\sigma TenantID=currentTenant\wedge TableID=currentTable(R))\}.$$

定义 2. *Data Table*.

表示数据表,以键值对方式存储租户业务数据,本模型中用关系表 *DataTable* 来存储.

$$DataTable=(TenantID,TableID,ColumnName,Value).$$

定义 3. *MappingIndex*.

表示映射表索引,用来存储索引列 A 与 B 的关系,本模型中该索引存储在缓存中.

$$MappingIndex=(TenantID,MappingID,Column1,Column2).$$

映射表索引在缓存中的实际存储如下:

$$MappingIndex(A,B)=\{node\}$$

$$node=(P0,(K_1,V_1,P_1),\cdots,(K_m,V_m,P_m))$$

$$0<i<m+1,$$

$$(K_i,V_i)\in\Pi_{A,B}(\sigma TenantID=currentTenant\wedge TableID=currentTable(R)),$$

$$^{*}p_i=node.$$

在第一次执行映射表索引更新时,需要花费时间和资源来为此做准备,为减少准备时间,本文采取了渐进式更新映射表的方式,为了记录每次更新所产生的索引切片(参见定义 6),本模型通过 *MappingColumn* 记录某一属性的切片列表,并通过 *MappingSet* 里的 *CurrentPivot* 记录了某一映

射的最新切片点.

定义 4.　*MappingPending*.

表示索引缓冲区,用来缓存待插入的索引列 A 与 B 的关系,等该数据需要被访问的时候,合并到 *MappingIndex* 中.

MappingPending=

(*TenantID*, *MappingID*,*Column*1,*Column*2).

为提高更新速度,索引缓冲区只有追加操作,不考虑有序性.

定义 5.　*MappingSet*(A).

表示 *MappingSet* 的关于属性 A 的一个子集,表示的是当前租户当前应用建立在 A 上的索引,描述了索引列 A 和其他列的两两关系.

$$MappingSet(A)=\{MappingIndex\ (A,X)\mid X\in \Pi_{columnName}(\sigma TenantID=currentTenant\wedge TableID=currentTable(columns))\}.$$

MappingSet(A)的缓存需求取决于所属 Table 的列的数目,即$|MappingSet(A)|=count(column)$,这将是一个很大的代价.

出于上述考虑,本文的 *MappingColumn*(A)在租户定制索引的时候建立,但是具体包含的 *MappingSet*(A,X)则根据具体的查询驱动建立,这样对于一些不经常用的数据就不会过早地占用资源.

定义 6.　*Slice*.

切片,是 *MappingIndex* 的一个子集,表示的是当前租户当前应用根据租户请求进行分片后形成的映射集合.

3　基于键值对模式的多租户索引维护策略

租户定制索引时,系统把索引结构映射到索引元数据中.基于索引元数据的映射,索引数据共享地存储在一个索引表中,首先屏蔽了多租户索引模式的差异,其次有效地解决了传统数据透视表数目过多导致性能下降的问题,最后能解决索引数据中标识性信息(如 *TableID*,*ColumnID*)重复存储的问题,节约了索引数据存储空间.

访问数据时,根据访问参数对索引数据进行切片,同一个切片内部的数据含义一致,解决了传统索引中索引难以理解及按租户标识进行过滤的问题.

关键过程 1.　Create Table.

用于在多租户数据库中创建表,本模型中把命令转化为针对元数据的操作.

insert into *tables*(*TenantID*,*TableID*,*Tablename*)

insert into *columns*

(*TenantID*,*TableID*,*ColumnName*,*DataType*,*Length*,*Null*)

关键过程 2.　Create Index.

表示在多租户数据库中创建映射表索引,需要根据租户对 table 的元数据描述,来创建各自的索引.

create index *idx* on *tablex*(A)

→create *MappingColumn* (A);

→insert into *MappingColumn* ((*TenantID*,*TableID*, A,*null*))

循环插入产生多个 *MappingID*:

insert into *MappingSet* (*TenantID*,*MappingID*, *TableID*,'*A*', *Column*2)

对于表 2 中的数据,假设租户 001 定制 A 为索引列,租户 002 定制 B 为索引列,则当前模型中实际存储中应该只有 *MappingColumn* 插入了索引信息,*MappingSet* 和 *MappingIndex* 都会到真正使用的时候才会插入信息.如表 3 所示.

表 3　*MappingColumn*

TenantID	*TableID*	*ColumnID*	*Pivots*
001	*R*	*A*	
002	*R*	*B*	

3.1　映射表索引维护策略

在事务性的 SaaS 应用中,业务数据持久化操作频繁,如果没有好的索引更新策略,在实时性要求较高的业务管理系统里是不可行的,本文提出了按需更新映射表索引的策略.业务数据更新时,只需要更新映射表索引缓冲区;业务数据查询时,首先根据元数据进行切片对齐,然后检查索引缓冲区是否有缓存数据需要合并,再按照索引进行查询,并返回结果.基于第 3 节的形式化定义,本文设计了如下的索引按需更新算法及相应的查询算法.

算法 1.　数据更新算法.

输入:insert into $R(A,B,C)$ *Values* ($v1$,$v2$,$v3$)

输出:null

1. 获取当前 *TenantID*;
2. 更新数据库中的业务数据;
3. 根据 *TenantID* 和 *ColumnID* 获取 *MappingColumn* 中对应的索引列;
   for 每一个索引列(*ColumnID*)
    for 每一个 *MappingSet*(*ColumnID*)
      查找 *MappingIndex* 及其 *MappingPending*;
      对 *MappingPending* 追加上对应 *column*1、*column*2 的映射关系;

next
next

该算法只对已经创建映射表索引的关系进行处理,如果插入时尚未建立索引,则会在首次查询的时候生成索引数据.

算法 2. 数据查询算法.

输入:select B from R where $v1<A<v2$
输出:符合条件的 B 的集合
1. 获取当前 $TenantID$;
2. 根据 $TenantID$ 获取 $MappingColumn$ (A)
2.1 若无,认为无索引,返回
3. if $MappingSet$ (A,B)=null, then
create $MappingSet$ (A,B)
4. 检查 $MappingIndex(A,B)$,根据 $MappingColumn(A)$ 的 $Pivots$ 列表进行对其补齐切片;
5. 检查 $MappingPending(A,B)$,若有,则调用算法 3;
6. 检查 $MappingColumn(A)$,看是否存在 $v1$,$v2$ 的切片点;
if 不存在 $v1$ then
对包含 $v1$ 的 $slice$ 按照 $v1$ 再次切片
检查 $v1$ 是否是 $slice$ 里边的值,
若不是,取 $v1$=本切片里的边界值;
检查 $MappingColumn(A)$中是否存在 $v1$,
若无,增加 $v1$,置 $CurrentPivot=v1$;
endif
if 不存在 $v2$ then
对包含 $v2$ 的 $slice$ 按照 $v2$ 再次切片
检查 $v2$ 是否是 $slice$ 里边的值,
若不是,取 $v2$=本切片里的边界值;
检查 $MappingColumn$ (A)中是否存在 $v2$,
若无,增加 $v2$,置 $CurrentPivot=v2$;
endif
7. 返回位于 $v1$,$v2$ 之间的 B 的值.

该算法以切片为数据访问的基本单位,能直接定位到要访问的 1～2 个切片,在进行一定次数的切片后,能有效地提高访问性能.

算法 3. 索引合并算法.

输入:$MappingIndex(A,B)$,$MappingPending(A,B)$
输出:合并以后的 $MappingIndex$ (A,B)
1. 遍历 $MappingIndex$ 的所有 $slice$,
从最后一个 $slice$ 开始,
首先将该 $slice$ 的第一项移到该 $slice$ 的最后一项后面,
判断是否应该落在该 $slice$ 中
2. 若属于,直接将该项插入到该 $slice$ 第一个位置,结束;
3. 若不属于,再判断前面的 $slice$,
若到达第一个 $slice$,
直接将该项添加到第一个 $slice$ 的第一项,结束.
4. 删除缓冲区里的 $MappingPending(A,B)$

3.2 映射表索引算法分析

本文提出的算法是正确而完备的.证明如下:

(1)对于任意索引列 x,必然在 $MappingCoulmn$ 中有记录;

(2)数据库更新时更新的数据在被查询时一定会进入索引中;

若更新时就已经存在映射(x,y),则所更新数据会保存到索引缓冲区中,当(x,y)被访问时,算法 2 会合并索引缓冲区到 $MappingIndex$ 中.

若还不存在映射(x,y),则当 x,y 被查询的时候,算法 2 会根据 $DataTable$ 创建 x,y 的映射关系,并根据 $MappingColumn$ 里的切片列表补齐切片工作,从而存在于 $MappingIndex$ 中.

(3)在查询时,由于各映射表在查询之前都进行了切片对齐工作,如要返回多个属性,直接根据切片进行选择以后的连串即为结果;

(4)在查询时,如果要根据多个条件进行查询,可以在连串中依次进行过滤,而不需要进行其它分支的查找.

综上所述,本算法是正确而完备的.

3.3 映射表索引算法执行结果

对于表 3 中建立的索引,一种可能的查询序列为 $A>2$,$A>6$ 索引切片序列为 2、6,结果如表 4 所示,$MappingColumn$ 中的 $pivots$ 也变为 2、6,此时的缓存中,可以很方便地返回 AB 的值,而无需进行多分支查找.在访问到 AC 的数据时,会按需建立 AC 的映射关系,适应了多租户数据模式差异及工作流程差异的需求.

表 4 *MappingColumn*

TenantID	*TableID*	*ColumnID*	*Pivots*
001	*R*	*A*	2,6
002	*R*	*B*	

表 5 *MappingSet*

TenantID	*TableID*	*MappingID*	*Column1*	*Column2*	*Current Pivot*
001	*R*	*M*01	*A*	*B*	6

表 6 *MappingIndex*($A>2$)

TenantID	*MappingID*	*Column1*	*Column2*
001	*M*01	1	3
001	*M*01	5	2
001	*M*01	8	9
001	*M*01	4	4

表 7 *MappingIndex*($A>6$)

TenantID	*MappingID*	*Column1*	*Column2*
001	*M*01	1	3
001	*M*01	5	2
001	*M*01	4	4
001	*M*01	8	9

4　实验结果与分析

本文建立了采用键值对共享模式存储的原型系统来验证映射表索引机制.本文中基于关系数据库,封装了一层元数据驱动的数据引擎,实现 SQL 重写和映射表索引的维护工作.本文的实验环境如下:

数据库服务器:HP 服务器,CPU 为 4×2.26,内存为 4GB,硬盘为 500GB.

应用服务器:联想,CPU 为 2×2.26,内存为 8GB,硬盘为 500GB.

客户机:联想,CPU 为 2×2.26,内存为 2GB,硬盘为 100GB.

为了更好地对比商业数据库和实验室系统的区别,本文基于 ORACLE 和 MYSQL 分别做了实验.

数据库版本 ORACLE 10.2.1.1(图 1～图 6),MYSQL5.1.47(图 7).

视图层每个关系模式具有 50 个属性,且指定其第一个属性为索引属性.

DataTable 中每个租户平均拥有 90000 条记录,DataTable 上有主码索引.

本文从以下几个方面作了实验验证:

(1)固定并发租户数目,比较无映射表方式、映射表索引方式的修改、修改后查询的性能.

(a)由 100 个租户共同发起 insert 和 delete,每个租户发起 100 次修改,参数均匀变换,以保证能均匀地访问各切片.例句如下:

insert into R (A,B,C) *Values* ('4'.'5','30');

insert into R (A,B,C) *Values* ('1800'.'200','1500');

insert into R (A,B,C) *Values* ('2500'.'100','500').

实验效果(图 3)表明按需更新和即时更新比普通更新的情况要稍微差一些,但按需更新比即时更新要好,这是符合实验情况的,并且也是可以接受的.

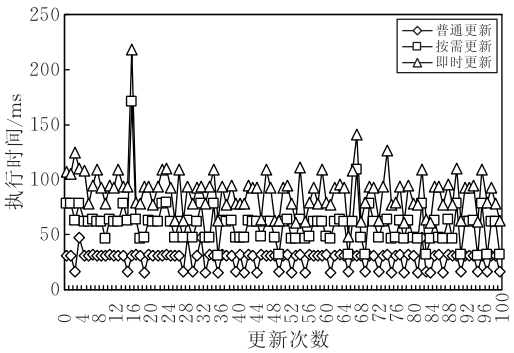

图 3　按更新次数比较更新代价

即时更新索引方式和按需更新索引方式,除了更新数据还要更新映射表,在原有代价基础上,增加了映射表索引的维护代价,分别是 100ms 和 50ms 左右.而在普通更新的方式下,无需维护映射表索引,也就没有额外的维护代价,因此维护代价仅包含插入数据的时间及建立主码的时间,一般是 25ms 左右.相应在图 4 中,能清晰地看出其查询代价一般在 80ms 左右.在应用中的读操作通常远远大于写操作,因此本文认为在此耗费的代价是可以接受的.

(b)由 100 个租户共同发起查询,每个租户发起 100 次查询,查询参数在业务数据的最小值和最大值之间均匀变换,比较更新后对查询的影响.

实验效果(图 4)表明在查询时按需更新索引方式和即时更新索引方式都要优于普通索引方式,按需方式与即时方式差别不是很大.

普通查询则因为无法实现隔离的问题,查询代价远高于本文所提出的算法.按需方式间断出现的几个较高代价是因为索引合并.索引缓冲区中的数据在被访问的时候,要合并到索引切片中,这需要一些额外代价,即时更新方式是在维护数据的时候同步进行的(参见图 3).从实验方案及效果来分析,这正好体现了索引缓冲的优势,它把每次更新操作转化成批量操作,降低了维护代价.

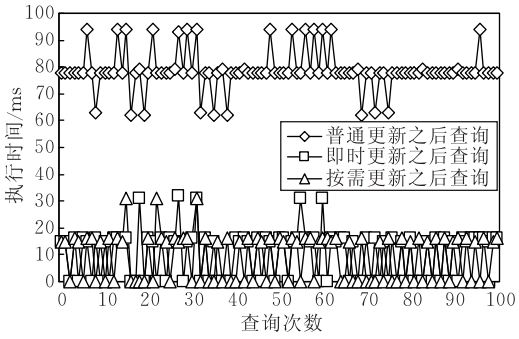

图 4　按次数比较查询性能

根据以上实验,可以看出使用映射表索引方式可以提供较好的性能,为了更好地判定租户数目对本维护策略的影响,本文模拟了并发租户数量变化的情况.

(2)固定按需更新映射表方式,比较不同数量的租户并发执行的性能.

由不同数量的租户并行各种业务,每个租户发起 100 次查询和修改,查询参数在业务数据的最小值和最大值之间均匀变换,更新参数任意均匀变化.查询和更新交替执行.

图 5 和图 6 分别为 ORACLE 和 MYSQL 的实

验效果,总体趋势一致,在性能上稍有差异,此差异是由于访问参数随机分布造成的,与数据库及数据引擎没有关系.实验效果表明:

(1)本方案和基于的数据库没有关系,具有较好的迁移性.这是因为本方案并没有直接在数据库中建立映射表,而是建立在缓存中,不依赖于数据库本身的能力.

(2)多租户数据库所支持的租户数量只受服务器缓存的限制,在缓存允许范围内,租户性能不受并发租户数量影响限制;在超过单机缓存负荷以后,需要寻求集群或者云等新的存储方式.

综上所述,本方案所提出的模型,实验效果理想,较好的解决了 SaaS 应用索引的隔离及定制问题.

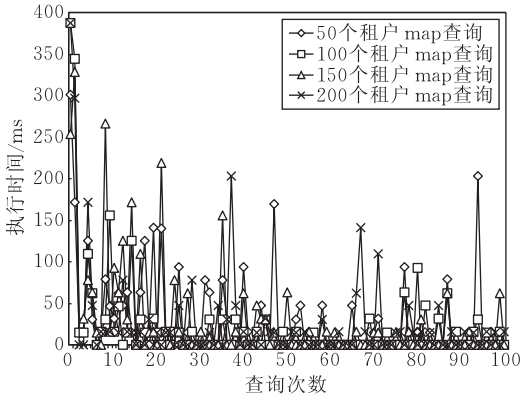

图 5 按并发租户数量进行比较(ORACLE)

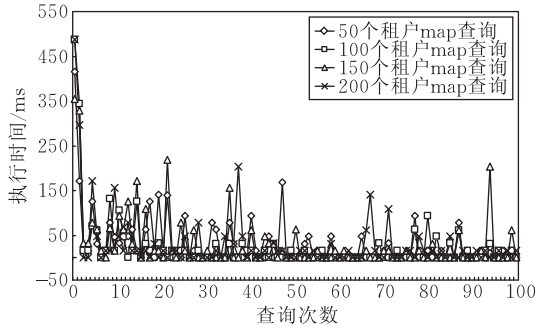

图 6 按并发租户数量进行比较(MYSQL)

5 总结与展望

本文基于键值对模式的多租户数据库存储机制,在 ORACLE 数据库中引入了映射表来充当索引以及元组重构的基础,同时提出了映射表的维护策略来降低映射表的维护代价,能有效地提高访问性能.本文根据租户信息创建共享模式的映射表索引,然后根据各自的数据访问请求,由数据引擎筛选索引切片,渐进式地对索引进行更新,获取较高的性能.本文论证了该索引方式的正确性、完备性、存储需求、维护代价及可获取的性能的提高,并以实验证明了本文思想的正确性.在实验中,本文从租户角度、查询次数、数据量等多方面对比了各种方案的查询及维护代价,可以看出本方案具有较好的使用效果.映射表索引还有尚未解决的问题,比如随租户、索引个数、属性数目增长的空间需求,对于索引查准率和查全率更严格的论证,数据引擎和关系数据库的中间交互环节的进一步优化,还有很多的研究工作要做.在今后的研究中,将做以下几个方面的工作:

(1)进一步论证映射表索引的查准率和查全率;

(2)研究通过索引进行关系重构的问题;

(3)研究关系中属性较多时候的索引引用技术以进一步提高索引性能.

参 考 文 献

[1] Weissman Craig D. The design of the Force.com multitenant Internet application development platform//Proceedings of the SIGMOD. Providence, Rhode Island, USA, 2009: 889-896

[2] Chong Frederick, Carraro Gianpaolo. Architecture strategies for catching the long tail. http://msdn2.microsoft.com/zh-cn/architecture/aa479069.aspx, 2006

[3] Chang Zhong-Zuo, Xu Yue, Dai Gang. The multi-tenant data architecture study based on the SaaS model for the public service platform. Computer Systems & Applications, 2008, (2): 7-11(in Chinese)
(昌中作,徐悦,戴钢. 基于 SaaS 模式公共服务平台多用户数据结构的研究. 计算机系统应用, 2008, (2): 7-11)

[4] Zhang Xiangyu, Ai Jing, Wang Zhongyuan, Lu Jiaheng, Meng Xiaofeng. An efficient multi-dimensional index for cloud data management//Proceedings of the CloudDB. Hong Kong, China, 2009: 17-24

[5] Wang Zhi-Hu, Guo Chang-Jie, Gao Bo, Sun Wei, Zhang Zhen, An W H. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing//Proceedings of the IEEE International Conference on e_Business Engineering. Piscataway, United States, IEEE CS, 2008: 94-101

[6] Aulbach Stefan, Grust Torsten, Jacobs Dean, Kemper Alfons, Rittinger Jan. Multi-tenant databases for software as a service: Schema-mapping techniques//Proceedings of the SIGMOD. Vancouver, BC, Canada, 2008: 1195-1206

[7] Hacigumus Hakan, Mehrotra Sharad, Iyer Balakrishna R. Providing database as a service//Proceedings of the 18th International Conference on Data Engineering. San Jose, California, USA, 2002: 29-38

[8] Kong Lanju, Li Qingzhong. An investigative approach on improving self service capabilities using XML//Proceedings of the 4th International Conference on Natural Computation.

Jinan, Shandong, China, 2008: 463-466

[9] Aulbach Stefan, Jacobs Dean, Alfons Kemper, Michael Seibold. A comparison of flexible schemas for software as a service//Proceedings of the SIGMOD. Providence, Rhode Island, USA, 2009: 881-888

[10] Abadi Daniel J, Madden Samuel R, Hachem Nabil. Column-stores vs. row-stores: How different are they really? //Proceedings of the SIGMOD. Vancouver, BC, Canada, 2008: 967-980

[11] Stonebraker M et al. C-store: A column oriented dbms//Proceedings of the VLDB. Trondheim, Norway, 2005: 553-564

[12] Abadi D et al. Materialization strategies in a column-oriented dbms//Proceedings of the ICDE. Istanbul, Turkey, 2007: 466-475

[13] Abadi D et al. Integrating compression and execution in column-oriented database systems//Proceedings of the SIGMOD, Chicago, Illinois, USA, 2006: 671-682

[14] Kersten M, Manegold S. Cracking the database sore//Proceedings of the CIDR. Asilomar, CA, USA, 2005: 213-224

[15] Idreos S, Kersten M, Manegold S. Database cracking//Proceedings of the CIDR. Asilomar, CA, USA, 2007: 68-78

KONG Lan-Ju, born in 1978, Ph. D. candidate, lecturer. Her research interests include database and semi-structured data management.

LI Qing-Zhong, born in 1965, Ph. D., professor, Ph. D. supervisor. His research interests include large-scale network data management and Web data integration.

SHI Yu-Liang, born in 1978, Ph. D., lecturer. His research interests include services computing and trusted computing.

WANG Xue, born in 1987, M. S. candidate. His research interests focus on database.

Background

A multi-tenant database system for Software as a Service should offer schemas that are flexible in that they can be extended for different versions of the application and dynamically modified while the system is on-line.

There are five techniques until now. In three of these techniques, the database "owns" the schema in that its structure is explicitly defined in DDL. Included here is the commonly used mapping where each tenant is given their own private tables, which we take as the baseline, and a mapping that employs Sparse Columns in Microsoft SQL Server.

These techniques perform well, however they offer only limited support for schema evolution in the presence of existing data. Moreover they do not scale beyond a certain level. In the other two techniques, the application "owns" the schema in that it is mapped into generic structures in the database. Included here are XML in DB2 and Pivot Tables in HBase. These techniques give the application complete control over schema evolution, however they can produce a significant decrease in performance. The authors conclude that the ideal database for SaaS has not yet been developed and offer some suggestions as to how it should be designed.

In order to excellently support SaaS application, multi-tenant database system needs to meet the tenants requirement of isolation and on-demand customization, and then to choose data storage and index mechanism that supporting isolation and flexibility. An effective way is to store data including the key-values based on meta-data. Based on relational database, this paper proposes a meta-driven indexing mechanism and according to tenant customization, the model constructs respective index metadata for business data of the tenants; while the index maintenance strategies are given and in response to tenant access requests, the model forms and slicing the index data. In this paper, detailed experimental results show that this program works with good performance under normal conditions.

The research is supported by the National Natural Science Foundation of China under Grant Nos. 90818001 and 61003253, the National Key Technologies R&D Program under Grant Nos. 2009BAH44B02 and 2009BAH44B04, the Natural Science Foundation of Shandong Province of China under Grant Nos. ZR2010FQ026, 2009ZRB019YT, 2009ZRB019RW, and Y2007G38, the Key Technology R&D Program of Shandong Province under Grant Nos. 2010GGX10105 and 2009GG10001002, the Independent Innovation Foundation of Shandong University under Grant No. 2009TS030.