计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(12):210−215 [doi: 10.15888/j.cnki.csa.006689]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

稀疏表与块表结合的多租户共享存储模型①

刘　彬[1], 程　凯[2], 于　杰[3]

[1](金诚信矿业管理股份有限公司, 北京 100044)
[2](北京宸控科技有限公司, 北京 102200)
[3](北京市新媒体技师学院, 北京 102200)
通讯作者: 程　凯, E-mail: chengkai@bjkcst.com

摘　要: 针对传统单稀疏表存储模型存储空间利用率低以及块表存储模型重构租户逻辑关系时连接次数多的问题, 提出了稀疏表与块表结合的存储模型. 该存储模型中将租户基于 SaaS 应用服务商提供的逻辑表上定制的属性及自定义的逻辑表中的属性映射到列数合适的稀疏表中存储, 而将租户一部分常见数据类型的自定义的属性存储到块表中, 以此避免因扩展字段的列数超过稀疏表列数导致的数据迁移问题, 最后通过定义查询重写器给出了从租户逻辑 SQL 请求到物理 SQL 请求的转换. 实验结果表明, 该存储模型在存储空间利用率及查询效率上相比传统的稀疏表存储模型都有所提升.

关键词: 多租户; 存储模型; 稀疏表; 块表; 云计算

引用格式: 刘彬,程凯,于杰.稀疏表与块表结合的多租户共享存储模型.计算机系统应用,2018,27(12):210–215. http://www.c-s-a.org.cn/1003-3254/6689.html

Multi-Tenant Shared Storage Model Combining Sparse Tables and Block Table

LIU Bin[1], CHENG Kai[2], YU Jie[3]

[1](JCHX Mining Management Co. Ltd., Beijing 100044, China)
[2](Beijing KingKong Science &Technology Co. Ltd., Beijing 102200, China)
[3](Beijing New Media Technical College, Beijing 102200, China)

Abstract: Aiming at the low utilization of storage space in traditional single-sparse storage model and the problem of more connections in reconstructing the logical relationship of tenants in the block table storage model, the combination of sparse tables and block table storage model is put forward. In the storage model, the attributes in the logic tables provided by the SaaS providers and the attributes in the custom logical tables are mapped into the corresponding sparse tables and the custom attributes of a part of the tenant common data type are stored in the block table, so as to avoid the problem of data migration caused by the number of extended columns exceeding the number of sparse tables. Finally, the query conversion and query efficiency are optimized through the query rewriter. The experimental results show that the storage model improves storage space utilization and query efficiency compared with the traditional sparse table storage model.

Key words: multi-tenant; storage model; sparse table; chunk table; cloud computing

随着云计算的发展及应用软件的成熟, 软件即服务 (Software as a Service, SaaS)[1]作为云计算的一种应用形式越来越受到重视. 多租户数据架构是搭建 SaaS 应用平台的关键技术之一, 不仅需要在数据库层面实现租户之间数据的隔离[2], 还要满足租户的定制需求.

目前几种典型的多租户共享存储模型, 包括透视表、稀疏表、块表及块折叠, 都能保障租户数据的隔离性和可定制性的需求, 但仍存在各自的不足. 例如,

① 收稿时间: 2018-05-06; 修改时间: 2018-06-04, 2018-06-15; 采用时间: 2018-06-22; csa 在线出版时间: 2018-12-03

在透视表存储模型中, 租户数据被拆成键值对形式的元组垂直存储, 这使得重构租户逻辑关系需要做大量连接操作, 重构一个 n 列的表需要做 n–1 次连接; 在稀疏表存储模型中, 不同租户的逻辑表都被映射到一个列数很大的表中, 例如 Salesforce.com 的数据表有 500 列[3], 从而会导致表中包含大量空值, 存储空间利用率低; 块表存储模型在透视表存储模型的基础上进行了改进, 一个块中包含若干不同数据类型的字段, 但在租户数据类型相同的字段数量多的情况下, 重构租户逻辑表仍然需要大量连接; 块折叠存储模型在块表存储模型的基础上做了进一步改进, 通过垂直划分将租户逻辑表中共有的属性存储在基本表中, 而剩余的属性仍然是采用块表存储, 并没有解决块表存储模型中存在的问题. 综上所述, 目前并没有完全成熟的多租户数据库架构[4], 因此如何提高多租户数据库性能仍然是值得研究的问题.

本文提出一种稀疏表与块表结合的存储模型, 旨在提升稀疏表存储模型的存储空间利用率, 并结合块表存储模型以更好地满足不同租户的个性定制需求.

1 相关工作

针对 SaaS 应用场景下的多租户数据共享存储模型, 国内外学术界的研究者们已经做过大量研究. 文献[5]中提出了一种基于 SaaS 化多租户数据进行分区的模型和策略, 使得多租户共享存储模型能够实现由单节点向多节点的扩展. 文献[6]中提出了一种在关系数据库中集成 xml 的方案, 即将 xml 数据类型的文档插入到数据库的大对象字段中, 但对 xml 文档进行解析的过程较耗费时间, 从而影响数据库整体性能. 文献[7]中提出了一种多稀疏表的存储模型, 即按照租户逻辑表的列数将其映射到不同列数的稀疏表中, 较之传统单稀疏表存储模型减少了空值存储, 提升了存储空间利用率, 但只是通过预留列的方式满足租户对逻辑表的扩展需求, 而当扩展列数超出稀疏表列数时仍需要进行大量的数据迁移, 极大影响数据库性能. 文献[8]从缓存的角度提升多租户数据库查询性能, 提出了一种基于块折叠存储模型的缓存管理机制. 文献[9]中针对传统块折叠存储模型提出了一种多级块折叠存储模型, 较之传统块折叠存储模型, 提升了查询性能和存储空间利用率. 文献[10]中针对多租户数据库一般定制下的自适应数据模式和高可定制下的个性化数据模式分别进行了设计, 并对基于相变存储器的数据库索引作了设计, 从而达到改善 SaaS 化系统的存储开销、可扩展性和系统性能的目的. 综上所述, 现有的多租户共享存储模型重构租户逻辑表仍需要较多连接操作, 其存储空间利用率和查询访问效率较低, 需进一步改善.

本文提出一种稀疏表与块表结合的存储模型, 将租户属性划分为在应用服务商提供的逻辑表的基础之上定制的属性和租户扩展的自定义属性, 然后分别映射到一组列数不同的稀疏表中及块表中, 较之传统稀疏表存储模型, 在存储空间利用率及查询效率上都有所改善.

2 存储模型定义

2.1 租户逻辑视图

对于多租户共享存储模型来说, 尽管在实际的物理存储结构中, 各个租户的数据都被存储在同样的数据库及数据表中, 但是租户之间不会感觉到其他租户的存在, 更不会访问其他租户的数据. 租户可以在应用服务商提供的逻辑表的基础之上定制各自需要的属性, 也可以往逻辑表中添加新的自定义的属性, 甚至还可以自定义新的逻辑表. 如图 1 所示, 各个租户可以针对 SaaS 平台中的各个应用, 通过定制生成各自私有的逻辑表, 之后就可以针对逻辑表做查询和访问操作, 而不用关心底层数据库中表的结构.

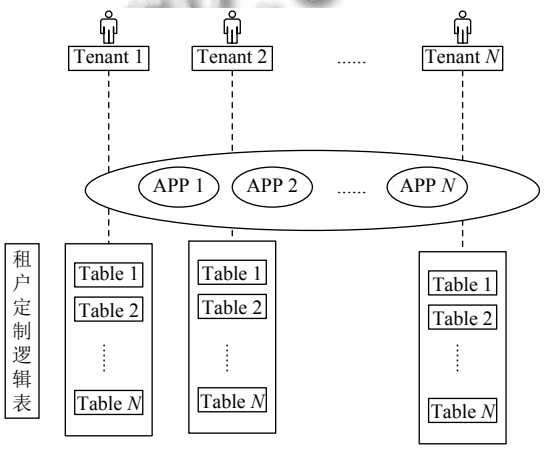

图 1 租户逻辑定制示意图

2.2 物理存储图

物理存储是指数据库中表的组织形式, 对于上层租户来说是透明的. 数据库中包括三种类型的表, 分别是: 稀疏表、块表及元数据表.

块表用于存储租户自定义字段, 当租户在逻辑表中扩展自定义字段时, 若该字段数据类型与块表中包含的某个字段的数据类型一致, 则将该字段映射到块表中相应的字段存储. 因此对于块表中字段的选取应结合具体 SaaS 应用的业务特征, 进而选择最常用的几种数据类型.

稀疏表用于存储租户在逻辑表上定制的字段、字段的数据类型不包含在块表中的租户自定义字段以及租户自定义的逻辑表中的字段. 本文在传统稀疏表的基础上添加了 row 字段, 表示租户逻辑表中数据记录的行号, 并作为重构租户逻辑关系时稀疏表和块表的连接条件之一.

元数据表包括对稀疏表进行描述的表及对块表进行描述的表. 其中, 对稀疏表进行描述的表存储了稀疏表与逻辑表之间的对应关系以及各个稀疏表的列数. 而对块表进行描述的表存储了块表与逻辑表之间的对应关系以及块表的表名和表中字段的数据类型.

2.3 模式映射

模式映射是建立逻辑表与物理表之间的映射关系, 即将逻辑表中的字段对应到物理表中. 首先要合理划分一组不同列数的稀疏表并给一组块表设置合适的字段. 针对不同列数的稀疏表的划分, 本文提出的策略是, 先统计应用服务商为租户提供的各个逻辑表的字段个数, 并保存到集合 $s_l\{s_{l_1}, s_{l_2}, \cdots, s_{l_n}\}$中, 字段个数相同的只计一次; 再依次取集合 s_l 中的值加 α 作为列数来创建相应列数的稀疏表, α 表示预留列数, 用于存储租户扩展字段. 针对块表中字段的设置, 由于稀疏表中预留的列数有限, 且租户的定制不是一次性完成的, 当租户向已有的逻辑表中添加新的自定义的字段时, 应让更多的扩展列映射存储到块表中, 少数的扩展列存储到稀疏表中, 这样可以降低稀疏表列数溢出的概率, 因此可以根据 SaaS 应用的业务特征来选择扩展字段可能的数据类型.

模式映射的具体过程为: 当租户定制逻辑表时, 先判断是否为租户自定义的逻辑表, 若是自定义的, 则先获取每个稀疏表的列数, 再根据二分查找法找到列数大于且最接近的该逻辑表字段数的稀疏表, 再在相应的元数据表中存储映射关系; 若不是租户自定义的逻辑表, 则需要进一步判断该逻辑表中是否包含租户自定义的字段, 若不包含则映射到相应的稀疏表中, 若包含自定义的字段, 则通过查询元数据表, 判断该字段的数据类型是否与块表中的某个字段数据类型一致, 再根据判断的结果将其映射到相应的稀疏表或者块表中. 由逻辑表映射到物理表的过程如图 2 所示.

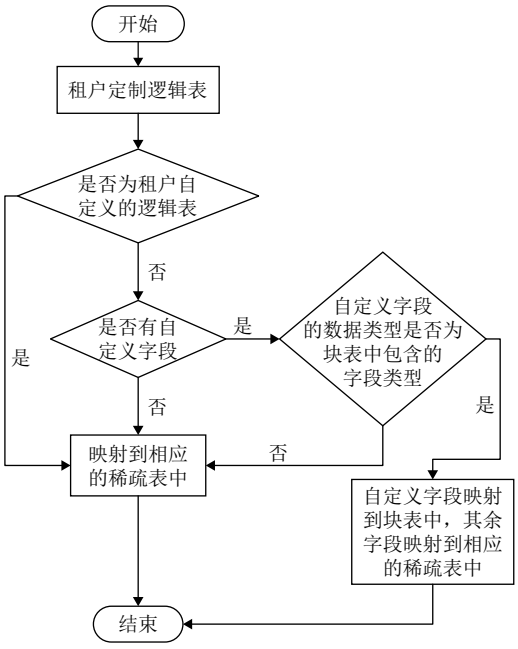

图 2 模式映射过程

2.4 合理性分析

要验证稀疏表与块表结合的共享存储模型的合理性, 只需要证明该共享存储模型与传统关系模型是等价关系即可. 证明的过程大致可分为两个步骤:

第一步要证明传统关系模型可以等价转化为稀疏表与块表结合的存储模型, 即对于租户定制的任意一个逻辑表, 假设为 R, 对于 R 中的任意属性 V, 其都可以被映射存储到稀疏表与块表结合的存储模型中. 首先若 V 是应用服务商提供的属性, 则将 V 映射存储到相应的稀疏表中; 其次若 V 是租户自定义的属性, 则根据该属性的数据类型分为两种情况, 若块表中包含与 V 的数据类型相同的字段, 则将 V 映射存储到块表中, 否则将 V 映射存储到相应的稀疏表中; 最后若 V 是租户自定义的逻辑表中的属性, 则将 V 映射存储到相应的稀疏表中.

第二步要证明稀疏表与块表结合的存储模型可以等价转化为传统关系模型, 即通过选择、连接及投影三种标准关系运算重构租户逻辑关系. 首先对于租户自定义的逻辑关系, 假设为租户 8 的 device 表, 先根据 md_sparse 表中的元数据信息找到对应的稀疏表, 假

设为 sparse_i 表, 然后对 sparse_i 表做选择和投影操作形成视图 X, 关系代数表示为:

$$X=\Pi_{col1,col2,\cdots,coli}(\sigma_{tid=8\wedge tableName='device'}(sparse\_i)) \quad (1)$$

最后对 X 进行更名操作, 即形成租户逻辑关系 R, 关系代数表示为:

$$R=\rho_{R(col1,col2,\cdots,coli)}(X) \quad (2)$$

其中, i=1,2, …, n; 其次对于应用服务商提供给租户的逻辑关系, 假设为租户 8 的 user 表, 并假设租户 8 在 user 表中的定制属性存储在稀疏表 sparse_i 中, 先对 chunktable 表做自身连接及投影操作形成视图 X, 关系代数表示为:

$$X=\Pi_{col1,col2,\cdots,coli}(\sigma_{chunk=0}(chunktable)\infty\sigma_{chunk=1}(chunktable)\infty\cdots\infty\sigma_{chunk=i}(chunktable)) \quad (3)$$

再对 X 和相应的稀疏表做自然连接及投影操作形成视图 Y, 关系代数表示为:

$$Y=\Pi_{col1,col2,\cdots,coli}(X\infty sparse\_i) \quad (4)$$

最后对 Y 做更名操作形成租户逻辑关系 R, 关系代数表示为:

$$R=\rho_{R(col1,col2,\cdots,coli)}(Y) \quad (5)$$

上述公式中, i=1, 2, …, n.

通过以上两个步骤, 即可证明本文提出的稀疏表与块表结合的存储模型与传统关系模型等价, 即租户所有基于传统关系模型的操作都可以转化为基于稀疏表与块表结合的存储模型完成.

3 存储分析及查询转换

3.1 存储对比分析

对于传统的稀疏表存储模型和块表存储模型来说, 影响存储空间利用率的最大因素是表中的空值. 空值既包括该属性值为空, 还包括租户未定制而产生的空值. 由于各个租户逻辑表的结构各异且稀疏表列数大, 稀疏表中必然存在租户未定制该列而产生的空值. 与传统稀疏表存储模型中只有一张宽度很大的稀疏表相比, 稀疏表与块表结合的存储模型中划分了一组不同列数的稀疏表, 进而将租户逻辑表映射到列数与之接近的稀疏表中, 因此稀疏表中的空值大量减少. 而块表中用于映射租户逻辑字段的列数很少, 因此块表中的空值相对于传统稀疏表存储模型来说也很少.

假设平台中有 n 个租户, 且一共定制了 m 个逻辑表 (Tl_1, Tl_2, …, Tl_m), Tl_i 的列数为 Cl_i, 行数为 Rl_i, Cl_i 列中有 E_i 列自定义的字段存储在块表中. 设根据逻辑表的字段个数划分了 k 个列数不同的稀疏表 (TS_1, TS_2, …, TS_k), Ts_i 的列数为 Cs_i, 且有 $Cs_1 < Cs_2 < \cdots < Cs_k$, Ts_i 中存储了 L_i 个租户逻辑表. 设块表的列数为 T_c, 且块表中存储了 L_n 个逻辑表中的自定义字段. 则可得到本文中存储模型的存储利用率 ρ 为:

$$\rho=\frac{\sum_{i=1}^{m}(Cl_i*Rl_i)}{\sum_{i=1}^{k}\left(Cs_i*\sum_{j=1}^{L_i}Rl_j\right)+Tc*\sum_{j=1}^{L_n}Rl_j} \quad (6)$$

设传统稀疏表存储模型中表的列数为 Max, 则该存储模型的存储利用率 ρ, 为:

$$\rho'=\frac{\sum_{i=1}^{m}(Cl_i*Rl_i)}{Max*\sum_{i=1}^{m}Rl_i} \quad (7)$$

将式 (6) 与式 (7) 相减, 可得到两种存储模型下的存储空间利用率的差值, 将该值简化后的分子表示为:

$$\rho_{diff}=Max*\sum_{i=1}^{m}Rl_i-\left(\sum_{i=1}^{k}\left(Cs_i*\sum_{j=1}^{L_i}Rl_j\right)+Tc*\sum_{j=1}^{L_n}Rl_j\right) \quad (8)$$

由式 (8) 可看出当一组稀疏表的存储容量之和加上块表的容量等于传统稀疏表的容量时, 两种存储模型下的存储空间利用率才相等. 然而块表中的列数一般会比宽度最小的稀疏表列数还少, 且通过换算可以得到:

$$\sum_{i=1}^{k}\left(Cs_i*\sum_{j=1}^{L_i}Rl_j\right)<\sum_{i=1}^{k}\left(Max*\sum_{j=1}^{L_i}Rl_j\right)=Max*\sum_{i=1}^{m}Rl_i \quad (9)$$

因此我们可以得出稀疏表与块表结合的存储模型较之传统稀疏表存储模型在存储空间利用率上有所改善.

3.2 查询转换

由于数据库中实际的物理存储结构对租户来说是透明的, 且所有租户对于数据的查询和访问操作都是针对其各自私有的逻辑表进行的, 因此需要通过程序中的查询重写器将租户针对逻辑表发起的逻辑 SQL 请求转换为针对物理存储结构的 SQL 请求, 最后将结

果返回给上层租户.

查询转换的过程可分为如下四个步骤:

(1) 获取租户唯一标识, 并从逻辑 SQL 请求中获取逻辑表的表名、要查询的字段名及查询条件

(2) 查询元数据表, 并根据逻辑表的表名和租户唯一标识获取该逻辑表存在映射关系的物理表的表名, 再获取逻辑表中的字段与相应物理表中字段之间的对应关系

(3) 根据上一步骤中得到的逻辑表中字段与物理表中字段之间的对应关系来构建针对物理表进行查询的 SQL 语句, 并分别为该 SQL 语句中查询的所有物理表中的字段名设置别名, 且别名均为逻辑表中与之对应的字段名

(4) 重新改写逻辑 SQL 请求, 用上一步骤中得到的 SQL 语句来代替原始 SQL 语句中逻辑表的表名, 即接在 from 子句之后表名

经过上述四个步骤, 可将逻辑 SQL 请求转换为物理 SQL 请求, 从而完成查询访问操作.

4 实验结果与分析

4.1 实验环境配置

操作系统: 64 位 Windows 8.1; 处理器: Core i5-4210M @ 2.60 GHz 双核; 内存: 8 GB

数据库: MySQL 5.5.28

实验中通过程序生成了模拟的 100 个租户 user 表的数据, 然后将其分别映射到稀疏表与块表结合的存储模型及传统稀疏表存储模型的物理表中. 传统稀疏表用 1 张列数为 500 的大宽表存储所有租户的数据, 稀疏表与块表结合的存储模型采用了 5 张列数分别为 33、43、53、63、73 的稀疏表和 1 张列数为 10 的块表存储所有租户数据. 各个租户的 user 表中的字段个数为 15–34, 且每张 user 表中都有 1000 条记录.

4.2 结果分析

对于存储空间利用率的分析, 本文中采取的做法是分别将传统稀疏表存储模型和稀疏表与块表结合的存储模型下的数据表导出为 SQL 文件, 分别命名为“稀疏表与块表.sql”和“single_sparse.sql”. 比较两个文件的大小, 前者文件大小为 25.5 MB, 而后者文件大小为 292 MB, 由此可知稀疏表与块表结合的存储模型相比传统的单稀疏表存储模型显著地提升了存储空间利用率.

对于查询效率的分析, 本文中采取的做法是模拟多个租户同时发起查询, 每个租户发起 400 个查询请求, 然后计算不同并发查询线程个数的情况下, 租户的平均查询响应时间. 两种存储模型下的查询效率对比如图 3 所示.

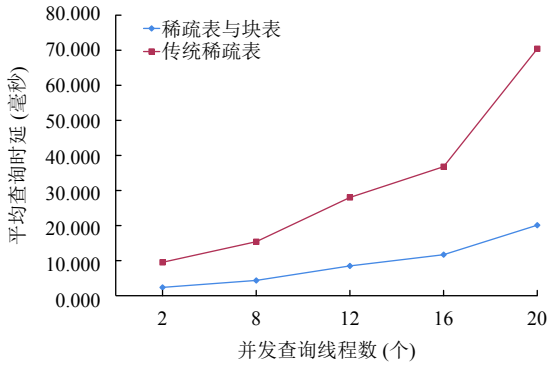

图 3 两种存储模型的查询效率对比

由图 3 可看出, 稀疏表与块表结合共享存储模型的平均查询时延小于传统稀疏表共享存储模型, 且随着并发查询线程数量的增加, 差值逐渐增大, 表明新提出的共享存储模型对查询访问效率进行了改善.

5 结论

针对传统单稀疏表存储模型存储空间利用率低及块表存储模型连接次数多的问题, 本文提出一种稀疏表与块表结合的存储模型. 该存储模型采用一组列数不同的稀疏表及块表共享存储多租户数据, 并构建相应的元数据表用于存储对稀疏表和块表进行描述的信息. 将租户基于服务商提供的逻辑表定制的属性及租户自定义的逻辑表中的属性映射到列数接近的稀疏表中, 而根据字段的数据类型将租户在逻辑表中自定义的属性映射到块表或者相应的稀疏表中. 通过存储分析及实验分析对比, 并从存储空间利用率, 查询效率及可定制性三个方面综合比较, 该存储模型性能优于传统的稀疏表存储模型.

参考文献

1 Elfatatry A, Layzell P. Software as a service: A negotiation perspective. Proceedings of the 26th Annual International Computer Software and Applications. Oxford, UK, UK. 2002. 501–506.

2 Zou LD, Li QZ, Kong LJ. Isolated storage of multi-tenant

data based on shared schema. Cybernetics and Information Technologies, 2016, 16(3): 91–103. [doi: 10.1515/cait-2016-0036]

3 Weissman CD, Bobrowski S. The design of the force.com multitenant internet application development platform. Proceedings of 2009 ACM SIGMOD International Conference on Management of Data. Providence, Rhode Island, USA. 2009. 889–896.

4 Aulbach S, Jacobs D, Kemper A, *et al*. A comparison of flexible schemas for software as a service. Proceedings of 2009 ACM SIGMOD International Conference on Management of Data. Providence, Rhode Island, USA. 2009. 881–888.

5 Li XN, Zhao JL, Ma YM, *et al*. A partition model and strategy based on the Stoer-Wagner algorithm for SaaS multi-tenant data. Soft Computing, 2017, 21(20): 6121–6132. [doi: 10.1007/s00500-016-2169-z]

6 Florescu D, Roquencourt I, Kossmann D. Storing and querying XML data using an RDMBS. Bulletin of the Technical Committee on Data Engineering, 1999, 22(3): 27–34.

7 Chen WL, Zhang SD, Kong LJ. A multiple sparse tables approach for multi-tenant data storage in SaaS. Proceedings of the 2nd International Conference on Industrial and Information Systems. Dalian, China. 2010. 413–416.

8 姚金成, 张世栋, 史玉良, 等. 基于 Chunk Folding 的多租户数据库缓存管理机制. 计算机学报, 2011, 34(12): 2319–2331.

9 何文哲, 范冰冰. 一种新型高效的多租户共享数据模型. 计算机应用与软件, 2017, 34(8): 66–71. [doi: 10.3969/j.issn.1000-386x.2017.08.012]

10 倪佳才. 多租户数据库设计的关键技术研究[博士学位论文]. 北京: 清华大学, 2015.