基于 Chord 的多租户索引机制研究

邹立达 李庆忠 孔兰菊 王振坤
(山东大学计算机科学与技术学院 济南 250101)
(山东省软件工程重点实验室 济南 250101)

摘 要 多租户数据管理是支持 SaaS 应用高效运行的重要组成部分.随着租户规模的不断扩大,多租户数据库需要云计算环境下的每个节点都存储并处理租户数据,因此适合利用对等结构(P2P)组织管理多租户数据.组织良好、易扩展的云中多租户索引机制是高效查询的关键.文中基于 P2P 结构对多租户索引机制展开研究,针对不同租户的索引易相互干扰、租户数据分布无序的问题,通过对 Chord 的映射方法进行改进将所有租户索引统一映射到一个标识符空间,给出的映射函数使单个租户索引可以隔离、保序地在空间分布.同时设计了各节点所负责的标识符空间范围的分配算法,使租户索引及数据可均衡、聚集地放置在各节点,从而使查询时既能通过对等结构避免性能瓶颈,也降低了数据传输成本.文中给出了多租户索引机制的动态维护策略,提出的标识符空间增倍方法使该索引机制能够适应租户数量与索引不断增加的应用场景.实验结果表明,该机制在租户规模较大时,与集中式索引相比查询时间至少可以节省 50%,吞吐量提高 1.5 倍.

关键词 多租户索引;P2P;Chord;多租户数据库;标识符空间;云计算
中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2016.00270

Indexing Mechanism of Multi-Tenant Database Based on Chord

ZOU Li-Da LI Qing-Zhong KONG Lan-Ju WANG Zhen-Kun
(*School of Computer Science and Technology, Shandong University, Jinan* 250101)
(*Shandong Provincial Key Laboratory of Software Engineering, Jinan* 250101)

Abstract Multi-tenant data management is an important part of supporting SaaS applications torunefficiently. With the continuous expansion of the tenants, multi-tenant database is deployed in cloud computing environment. Since every node in clouds stores and processes multi-tenant data, it needs a peer to peer (P2P) structure to organize multi-tenant data management system. In this case, a well-organized, scalable multi-tenant indexing mechanism in clouds is critical for efficient querying. In this paper we proposed a multi-tenant indexing mechanism supporting P2P structure. For solving the problems of mutual interference of multi-tenant indexes and disordered distribution of multi-tenant data, we improved the mapping approach of identifier space in Chord, which makes a single tenant index be stored in the cloud, in isolation and in order. Moreover, we designed an assignment algorithm of identifier space to evenly and clustered place multi-tenant indexes and data to nodes, which not only avoids bottleneck of querying performance through P2P structure, but also reduces data transmission cost. Last we presented a dynamic maintenance strategy of multi-tenant indexing mechanism to adapt to the scenario of increasing indexes and tenants by doubling identifier space. Through the extensive experiments, we demonstrate that the proposed mechanism outperforms centralized indexing mechanism with at least 50% of query time reduction and 1.5 times throughput increasement.

Keywords multi-tenant index; P2P; Chord; multi-tenant database; identifier space; cloud computing

收稿日期:2014-09-02;在线出版日期:2015-07-23.本课题得到国家自然科学基金(61572295,61303085)、山东省自然科学基金(ZR2013FQ014,ZR2014FM031)、山东省科技发展计划(2014GGX101047)和山东省自主创新专项项目(2015ZDJQ01002,2015ZDXX0201B03)资助.**邹立达**,男,1981 年生,博士研究生,主要研究方向为云数据管理与查询优化.E-mail:zoulida@163.com.**李庆忠**(通信作者),男,1965 年生,博士,教授,主要研究领域为大规模网络数据管理及 Web 数据集成.E-mail:lqz@sdu.edu.cn.**孔兰菊**,女,1978 年生,博士,讲师,主要研究方向为数据库、半结构化数据管理.**王振坤**,男,1991 年生,硕士研究生,主要研究方向为数据库.

1 引　言

随着 SaaS(Software as a Service)[1-2]应用规模的不断扩大，多租户数据管理成为 SaaS 应用快速开发和高效运行的重要基础. 数据处理资源被多个租户共享使用是多租户数据库的重要特征，对于共享方案，Chong 等人[3]提出了 3 种解决思路：独立数据库，即为每个租户建立一个数据库实例；共享数据库独立模式，即所有租户共享一个数据库实例，但每个租户拥有个性化定制的模式；共享数据库共享模式(又称共享模式)，即所有租户共享一个数据库实例及同一个数据模式，同时在表中通过租户标识区分数据所属的租户，此种方案的共享程度最高. 对于共享模式，Weissman 等人[1]提出了通用表共享方案，即将所有租户的业务数据存储到一个通用表中；Aulbach 等人[4]提出了 Chunk Folding 方案，即把租户的数据表垂直划分为多个块，将它们折叠存储至不同的 Chunk 表中；Ni 等人[5]则将租户共享的、关系依赖的属性存储在基表，其他的属性则存储在辅助表. Aulbach 等人[4,6]证明共享模式的系统性能最优，因此主流的多租户数据库多采用共享模式. 本文主要对共享模式多租户数据库的索引机制展开研究.

多租户数据库有租户数量多、单个租户数据量小但数据总量大的特点，集中式的数据处理结构易造成性能瓶颈[7]，因此需要云计算环境下的每个节点都能存储并处理数据，适合利用对等结构(P2P)组织管理多租户数据.

在云计算环境下为多租户数据建立良好的索引机制是多租户数据高效查询的关键. 目前，云中多租户索引主要存在以下两个问题：首先，在创建与维护索引时租户间存在相互干扰. 即在共享模式下，一个租户在某个属性上建立索引，会导致其他租户也在该属性上被动建立索引，即使其他租户没有索引需求. 其次，现有的多租户索引多以集中式索引为主，没有考虑云计算环境中多租户数据总量大的场景，易导致过多的查询集中在入口节点，使得查询性能下降.

P2P 协议[8]能较好地提供性能平衡与扩展的功能，适合利用 P2P 结构建立多租户数据索引. 在 P2P 协议中，Chord[9]协议能以较低的路由表维护代价提供高效的查询跳转效率，且索引映射结构简单易于改造，适用于多租户场景. 然而，基于 Chord 协议建立的多租户索引还存在以下挑战：(1) 索引分布的无序性增加了租户查询处理的数据传输成本. 多租户场景中每个租户的数据量小，且租户的查询具有隔离性，即只查询属于自己的索引与数据. 若一个租户的索引与数据只聚集地放置于一个或者少量节点，则在查询处理时，可避免或减少跨节点的数据传输. 然而，Chord 协议采用随机方法对索引进行标识符空间映射，属于一个租户的索引及数据散乱地分布在较多节点中，因此需要在节点间传输大量数据才能完成租户的查询处理功能；(2) 无法满足多租户数据索引对扩展性的需求. 随着多租户数据库投入使用，租户数量与数据量会以较快的速度增加[10]. Chord 协议无法保证新增的租户与原有租户隔离的分布，并且无法保证一个租户新增的索引放置到其租户数据所在的节点，因此会降低数据查询效率. 针对以上问题，需通过对 Chord 协议进行改进建立多租户索引机制，使其适用于多租户数据的特点，提高数据查询效率.

本文通过改进 Chord 映射方法对多租户索引进行组织管理，使建立的索引机制满足多租户数据库对隔离性、扩展性的需求. 本文通过将 Chord 标识符空间划分为若干个大小相同的区域，使每个区域对应一个租户，实现租户索引的隔离管理；设计标示符空间的分配算法，确定各节点所负责的空间范围，使单个租户的数据与索引在节点中聚集地分布与存储，以减少查询时数据传输的成本；给出多租户索引机制动态维护策略，使其适应租户与索引数量不断增加的应用场景. 由此本文建立了一个支持 P2P 结构的多租户索引机制，避免出现性能瓶颈的同时提供动态扩展性能，并通过实验验证了该机制在较大租户规模时查询效率、查询吞吐率有较好的表现.

本文第 2 节讨论多租户索引的相关研究工作及存在问题；第 3 节给出多租户索引机制的框架及相关定义；第 4 节改进 Chord 映射函数，并讨论索引条目在各节点的分布与组织；第 5 节给出索引机制的动态扩展方法；第 6 节对多租户索引机制进行实验验证；第 7 节给出本文的结论.

2 相关研究

对于多租户数据共享模式存储，索引管理是实现其高效查询的重要环节. 直接在共享模式存储的多租户数据上统一建立索引，将使得所有租户数据一起被索引，因此对于特定租户来说，在索引维护与查询时会产生严重的性能问题. 文献[1,6,11-12]针对多租户索引隔离问题展开了研究. Weissman 等

人[1]在其平台 Force.com 中使用一系列的数据透视表(Pivot Table)来存储租户的索引,有效地支撑了租户数据的高速访问,但随着租户数据量的增加,透视表占用的存储空间会暴涨,从而导致系统效率降低. M-Store[12]针对多租户共享索引查询效率不高的问题,提出了多分离索引技术. 其为不同租户单独建立索引,提供了索引可定制功能,但随着租户数量与索引数的增加,数据引擎则需管理大量的索引实例. Aulbach 等人[6,11]提出多种多租户数据共享存储模式,并介绍了 Chunk Table 与宽表租户索引建立方法,其根据租户是否有建立索引需求将租户数据分为两类,分别存储在两个 Table 中,在其中一个 Table 建立索引,另外一个则不建立索引,以较低的资源成本为租户索引提供了隔离性.

以上文献针对索引隔离问题给出了解决方案,但都没有考虑云计算环境的多租户索引结构,而大量的多租户数据需要在云中管理,并建立可扩展、支持大量并发查询的索引机制. 目前,根据组织方式云中的索引可分成两类:集中式结构和 P2P 结构. 集中式的索引方式[13-14]以一个节点作为查询入口,极易造成性能瓶颈[7]. P2P[15-17]结构的索引采用节点对等工作模式平衡负载,为解决性能瓶颈问题提供了一种途径,而且自组织的 P2P 结构允许节点自由加入与离开,为云中多租户数据索引管理提供了一种理论上可无限扩展的组织模型.

文献[18-19]基于 P2P 结构建立了可扩展和高吞吐量的索引机制,为云中的多租户索引提供了借鉴. Wu 等人[18]提出了 CG-index(Cloud Global index),即所有节点组成一个覆盖网络,每个节点上建立局部索引并选择索引信息发布到覆盖网络上,其根据查询模式自适应地选择可发布的 B 树节点,提高了查询效率. Chen 等人[19]提出了一种针对 DaaS 的云中类关系(DBMS-like)模型的索引机制,将索引保序地发布到 P2P 网络中,以支持范围查询与比较操作,为云中关系查询处理的索引建立提供了基础. 但是,在多租户环境中直接应用这些云索引技术既不能提供索引隔离的功能,也不能满足租户数量与数据量扩展的需求.

常见的 P2P 索引协议有 CAN[20]、BATON[21]、Pastry[22]、Chord[9]等. 相比其他 P2P 索引协议,Chord 能以较低路由表存储代价提供高效的查询跳转效率($O(\log N)$),且映射结构简单、易于改造与维护,适用于多租户场景. 因此,本文基于 Chord 协议建立多租户索引机制.

然而,直接使用 Chord 协议[9]建立索引机制无法满足多租户数据库对索引的特殊需求:首先,由于 Chord 协议采用随机方法对关键字位置及节点所负责的标识符空间范围进行映射,无法满足多租户数据库对索引在节点中隔离、聚集分布的需求. 隔离分布是指一个租户使用索引查询时,只能检索到自己的数据,而随机映射的索引条目使得租户在查询时,特别是在范围查询时,租户获得自己数据的同时也获得了其他租户的数据. 聚集分布是指一个租户的索引与数据尽可能分布于尽可能少的节点中,从而减少跨节点查询次数,而随机映射方法使得一个租户的索引分布于较多节点. 其次,多租户索引机制需要能动态地扩展. 多租户数量是动态增加的,由于 Chord 协议标识符空间的大小初始化后不能再更改,新增租户映射的位置会与原有租户产生冲突,破坏了租户间的隔离性. 除以上特殊需求之外,与单租户场景相同,多租户数据库也需要索引在标识符空间保序地映射及在节点间均衡地分布,而 Chord 协议[9]并不能较好地支持该需求. 因此需对 Chord 协议进行改进,使得建立的多租户索引在支持保序、均衡分布的前提下,实现各租户索引隔离、聚集地分布,并具有可扩展性.

3 云计算环境下的多租户索引框架

Chord 协议[9]可提供一种能在 P2P 中快速定位资源的方法,基本思想是将代表资源信息的关键字和服务器节点统一映射到一个标识符空间,并通过标识符空间定位资源位置. Chord 采用 SHA-1[23]将资源信息及节点随机映射到标识符空间.

然而,随机映射的方法可能将不同租户的索引条目映射到标识符空间的相同位置且不保证原有条目的顺序,所以不能适用于多租户数据索引隔离、保序的场景. 因此本文提出一种通过改进 Chord 映射方法的多租户索引机制(Multi-tenant Indexing Mechanism Based on Improved Chord Mapping Approach, MIMC)为多租户提供索引管理功能. MIMC 的组织结构如图 1 所示:多租户数据库的索引分布在各个局部节点中,且索引与其指向的数据放置在同一个局部节点;每个局部节点维护一个路由表,并利用 B+树建立本节点的局部索引;所有节点基于 Chord 协议形成多租户数据的云中索引机制;在查询时,先通过路由表找到索引条目所在节点,再通过局部索引得到租户数据.

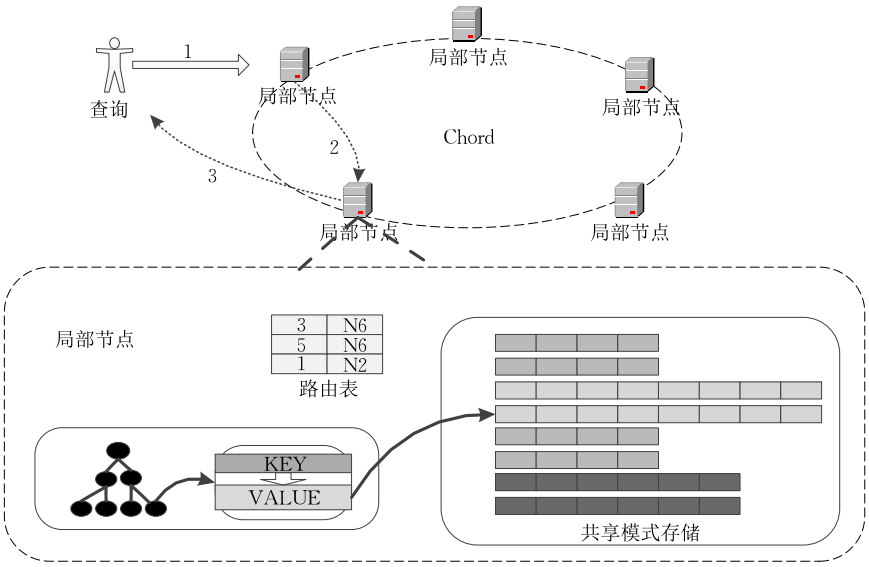

图 1　云中的多租户索引机制

为方便描述，相关定义如下.

定义 1.　租户. 租用多租户应用的组织或用户称为租户. 用 tn_i 表示，i 为租户编号，$0<i\leq I$，I 为租户数量.

定义 2.　数据对象. 用于描述 tn_i 某一业务对象的一条数据记录称为数据对象，表示为 rd_j. 用于描述相同业务对象的数据记录集合称为数据对象集合，表示为 $D_k=\bigcup\{rd_j\}$，k 为一个租户数据对象集合的编号.

定义 3.　多租户数据库. 存储与处理租户数据的系统称为多租户数据库. 多租户数据库以共享的方式被租户使用，数据对象集合以共享模式的方式存储，通过定制可满足租户对模式的不同需求，租户通过应用访问多租户数据库.

定义 4.　索引条目. 一个租户的索引可索引基本存储元素称为索引条目. 索引条目是以键值对形式进行管理的，用 $e_r:\langle k,v\rangle$ 表示一个索引条目，r 是索引条目的标识，k 表示索引列的属性值，v 是索引条目指向一个或多个数据对象的物理地址.

定义 5.　数据对象索引. 针对一个租户 tn_i 数据对象集合的某一属性，为加快数据对象集合查询速度而建立的索引条目集合称为数据对象索引，表示为 $\mathcal{I}_{i,j}$. 一个数据对象集合可有多个数据对象索引，即对于一个数据对象集合，可根据租户的需求在其不同属性上建立若干个数据对象索引. 数据对象索引表示为 $\mathcal{I}_{i,j}$，其中 i 为所隶属租户 tn_i 的编号，j 为 tn_i 建立的第 j 个数据对象索引，j 以自然数编号. $\mathcal{I}_{i,j}$ 是若干索引条目组成的集合，即 $\mathcal{I}_{i,j}=\bigcup\{e_r\}$.

定义 6.　租户索引集. 针对一个租户建立的数据对象索引集合称为租户索引集. tn_i 的租户索引集表示为 $\mathbb{k}_i=\{\bigcup_{j=0}^{J}\mathcal{I}_{i,j}\}$，$J$ 为租户建立数据对象索引的数量. 多租户数据库中所有的租户索引集合称为索引集，用 $\mathbb{I}$ 表示，那么 $\mathbb{I}=\{\bigcup_{i=0}^{I}\mathbb{k}_i\}$.

定义 7.　标识符空间(Identifier Space). 标识符空间是一致性哈希值[23]的范围，表示为 S，$S\in\{[0,2^m-1]\mid m$ 标识符空间的位数$\}$. S 的大小表示为 $|S|$，则 $|S|=2^m$. 对于任意一个 e_r，利用映射函数 F 可将 k 映射到 S 中. 设 k 映射到 S 的值表示为 k^s，即 $k^s=F(k)$.

通过定义映射函数可使所有的索引条目与局部节点通过哈希函数都映射到 S，从而得到局部节点与所负责管理的索引条目的对应关系. 所有的局部节点形成一个覆盖网络，因此 S 的每个值都被分配到某个局部节点中，那么索引条目则根据 k^s 的值分配到对应的局部节点. 每个局部节点负责一部分标识符空间，且所负责的标识符空间不重叠. 用 $R_h=[l_a,l_b)$ 表示局部节点 n_h 负责标识符空间范围，那么 $S=\{\bigcup R_h\mid n_h\in\mathbb{P}\}$，$\mathbb{P}$ 为局部节点集合，且 $\forall R_g,R_h$，必有 $R_g.l_a>R_h.l_b$ 或 $R_h.l_a>R_g.l_b$. 局部节点只对落入自己负责的标识符空间的索引条目建立局部索引.

定义 8.　路由表. 每个节点存储一个路由表，路由表由局部节点与其所负责的标识符空间范围的对应关系的记录组成. 对于 n_h，路由表有 m(与标识符空间的位数相同)个路由条目，其第 i 个路由条目为 $\langle x^s,n_x\rangle$，x^s 是 S 的一个位置且 $x^s=(l_a+2^{i-1})\bmod 2^m$，$n_x$ 是负责 x^s 的局部节点.

本文对索引集$\mathbb{I}$的索引条目统一管理.在初始化时,首先在云计算层基于 Chord 协议组织将索引条目及相关数据对象集合分布到各局部节点中;其次局部节点用 B+树组织所负责的索引条目,并管理路由表.在数据查询时,给定查询 k 值,通过 F 计算 k^s,查找路由表,可找到负责管理任意 e_r 的 n_h,并在局部节点得到 v,从而定位所需数据.

本文建立的支持 P2P 多租户索引机制以索引条目为粒度进行管理.首先,研究索引条目在局部节点中的分布及局部的组织方式.主要通过设计租户索引条目在 S 的映射函数确定索引条目在节点中的分布,使租户的索引条目隔离、聚集、均衡的分布与存储.其次,针对新增租户与索引的标识符空间耗尽问题,本文研究了索引条目动态映射方法.采用倍增$|S|$的方法,将新租户的索引条目映射到新增的空间中去,使其适应租户数量与索引不断增加的应用场景.

4 索引条目的组织

本文利用一个统一的映射方案对多租户数据库中所有的索引条目进行组织.对于云中的多租户索引,需先将索引条目按照一定规则分布到各节点中,再在局部节点中组织存储所负责的索引条目.

本小节首先给出索引条目在标识符空间 S 中具体的映射函数,此函数可保证索引条目的顺序不变且各租户索引相互隔离;其次给出各节点负责 S 范围的分配算法,此算法可保证一个租户的索引条目分布相对聚集且各节点分配索引条目的数量分布均衡;最后给出索引条目在局部节点的组织方法,该方法将索引条目及其所索引的数据存储在同节点,提高了索引效率.

4.1 索引条目在标识符空间映射

在基于 Chord 多租户索引机制中,所有索引条目都映射到一个 S 中,每个索引条目在 S 的位置是通过映射函数给出的.Chord 使用的 SHA-1 哈希函数不能满足多租户索引保序、隔离的映射,所以重新设计索引条目的映射函数.

本文设计映射函数 F 的思想如图 2 所示.首先,将标识符空间 S 划分为若干小区域,每个区域分配给一个租户索引集使用,称为租户区域,表示为 S_{TN}.其次,将每个租户区域进一步划分为若干更小的区域,称为数据对象索引区域,表示为 S_{IN}.一个 S_{IN} 分配给一个数据对象索引使用.再次,将属于一个数据对象索引的索引条目按照顺序依次映射到 S_{IN} 的位置中.

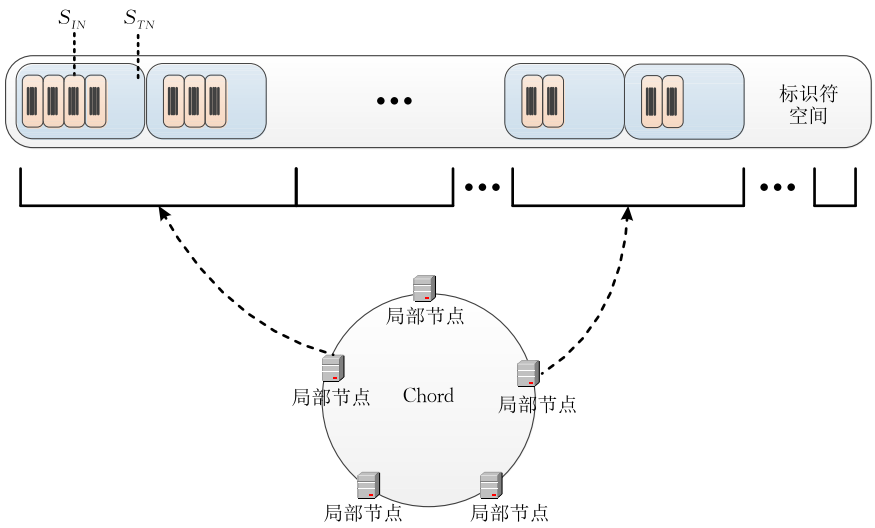

图 2 标识符空间的划分

现给出 $\mathcal{I}_{i,j}$ 任意一个索引条目 e_r:〈k,v〉在 S 的映射函数 F:

$$F(k)=|S_{TN}|\times(i-1)+|S_{IN}|\times(j-1)+|S_{IN}|\times(k-min_{i,j})/(max_{i,j}+1-min_{i,j}) \quad (1)$$

式中$|S_{TN}|$是S_{TN}的空间大小,$|S_{TN}|=|S|/Max_{TN}$,Max_{TN}是在 S 中可提供索引管理服务的租户数量.$|S_{IN}|$是S_{IN}空间大小,$|S_{IN}|=|S_{TN}|/Max_{IN}$,$Max_{IN}$是系统为每个租户可建立数据对象索引的最大数.$Max_{TN}$、$Max_{IN}$与$|S|$值的详细设置见第 6 节.由于 k 是某一数据对象的属性值,那么可知该数据对象所定义的数据类型的取值范围,函数(1)中 $min_{i,j}$ 与 $max_{i,j}$ 是 k 所属数据类型可取值范围的最小值与最大值.比

如，若 k 的数据类型为 MYSQL 的 BigInt，$min_{i,j}$ 与 $max_{i,j}$ 则分别为 0 与 $2^{64}-1$.

函数(1)给出 k 是数值型时的映射函数，k 也可以是字符串型、日期型、多键值索引等多种数据类型. 若为字符串型可利用可变长度文本串哈希[24]进行映射，若需支持范围查询则使用位置敏感哈希[25]映射，k 如果是多键值索引，可以利用 Z-Curve[19,26] 或者 Hilbert-Curve[27] 的方法将多维键值映射到一维空间.

由于所有数据对象索引都在相同大小的 $|S_{IN}|$ 中进行映射，对于取值范围较大的数据类型，即当 $max_{i,j}-min_{i,j}>|S_{IN}|$ 时，不同 k 值的索引条目可能映射到 S 中的相同位置，从而产生哈希冲突. 可采用链地址法解决冲突，链地址的数据结构可以是线性列表或者时间复杂度为 $O(\log n)$ 的自动平衡树[28].

函数(1)应能保证索引条目的位置性与隔离性. 以下给出相关定义.

定义 9.　位置性. 位置性是指属于同一数据对象索引 $\mathcal{I}_{i,j}$ 的索引条目 e_r 映射到标识符空间 S 中仍然保持原有顺序，即 $\forall e_a:\langle k_a,v_a\rangle\in\mathcal{I}_{i,j}$，$\forall e_b:\langle k_b,v_b\rangle\in\mathcal{I}_{i,j}$，if $k_a<k_b\Rightarrow F(k_a)<F(k_b)$.

定义 10.　隔离性. 隔离性是指属于不同 $\mathcal{I}_{i,j}$ 的 e_r 不会映射到 S 中的同一个位置. 也可表述为一个数据对象索引在空间 S 中的分布有着自己独立的空间，与其他数据对象索引的映射空间相互隔离. 形式化表示为 $\forall e_a:\langle k_a,v_a\rangle\in\mathcal{I}_{i,j}$，$\forall e_b:\langle k_b,v_b\rangle\in\mathcal{I}_{k,l}$，if $\mathcal{I}_{i,j}\neq\mathcal{I}_{k,l}\Rightarrow F(k_a)\neq F(k_b)$.

定理 1.　映射函数(1)具有位置性与隔离性.

证明.　根据函数(1)，在同一个 S_{IN}，映射函数是单调递增的，所以保证了 e_r 的位置性. 对于隔离性，当 $i=k$ 时，不失一般性，设 $l>j$，推出

$$F(k_b)-F(k_a)=$$

$$(l-j)\times|S_{IN}|+|S_{IN}|\times\frac{k_b-max_b}{(max_b+1-min_b)}-$$

$$|S_{IN}|\times\frac{k_a-max_a}{(max_a+1-min_a)}\geq$$

$$|S_{IN}|-|S_{IN}|\times\frac{min_a-max_a}{(max_a+1-min_a)}>0.$$

当 $i\neq k$ 时，不失一般性，设 $k>i$，推出

$$F(k_b)-F(k_a)=$$

$$(k-i)|S_{TN}|+(l-j)|S_{IN}|+|S_{IN}|\times$$

$$\frac{k_b-max_b}{(max_b+1-min_b)}-|S_{IN}|\times\frac{k_a-max_a}{(max_a+1-min_a)}\geq$$

$$|S_{TN}|-(Max_{TN}-1)|S_{IN}|-$$

$$|S_{IN}|\times\frac{k_a-max_a}{(max_a+1-min_a)}\geq$$

$$|S_{IN}|-|S_{IN}|\times\frac{min_a-max_a}{(max_a+1-min_a)}>0.$$

所以 $F(k_b)\neq F(k_a)$.　证毕.

每个 $\mathcal{I}$ 都对应映射到一个 $|S_{IN}|$，因此任意租户可以独立地建立与定制数据对象索引，不影响其他的租户使用，从而保证了数据对象索引的隔离. 同时，函数(1)的位置性可实现多租户数据的连接查询、范围查询等带有比较操作的查询处理. 新增索引条目可利用函数(1)动态插入，并保持位置性与隔离性.

4.2　索引条目在局部节点的分布

多租户数据库的索引条目及数据对象需分布到各局部节点，并在局部节点存储租户数据且建立局部索引. 首先，根据一定规则确定每个局部节点所负责的索引条目，该内容将在本小节论述，其次，每个局部节点先存储所负责的索引条目对应的数据对象集合，然后建立 B+树组织管理所负责的索引条目，此内容在 4.3 节论述.

根据 Chord 协议，一个局部节点通过哈希函数随机的分配到一部分标识符空间，通过哈希索引函数落入此空间的索引条目都被该节点存储与管理[9]. 然而随机分配局部节点所负责的标识符空间范围不能适应多租户索引的应用场景. 首先，随机分配会导致索引条目在节点中分布不均衡. 索引条目在节点中均衡分布是 P2P 结构高效查询的关键. 然而，映射到标识符空间的数据库索引并不是均匀分布的[29-30]. 特别是在多租户数据库中，由于不同数据对象索引所描述的数据对象集合的数据量大小及属性值分布是不同的，所以其包含的索引条目数量也不相同. 采用 Chord 的方式确定各节点负责的标识符空间范围，将导致数据分布严重不均. 其次，随机化分配导致一个租户的索引分布在多个节点中，增加查询时节点间数据传输的成本. 属于一个租户的索引条目应聚集地分布，即分配到尽可能少且相邻的节点上. 因此，需要设计标识符空间在节点中的分配方案，使索引条目在节点中均衡、聚集地分布.

不同于 Chord 通过哈希 IP 地址得到局部节点所负责的标识符空间的起始值与结束值，本文通过计算索引条目数量的方法得到局部节点 n_h 应负责的标识符空间 $R_h=[l_a,l_b)$. 为保证索引条目在各节

点分布数量的均衡化,在索引条目初始分配时,计算每个节点应分配的索引条目数量的平均值 $\bar{p}$,再按照各索引条目的 k^s 大小依次分配到各个节点,使每个局部节点负责的索引条目数量为 $\bar{p}$. 算法 1 给出 S 在节点中的分配方法.

算法 1. 标识符空间在每个节点中的分配.

输入:$\mathbb{I}$,$\mathbb{I}$ 中索引条目的数量 $|\mathbb{I}|$,局部节点数量 $|N|$

输出:每个局部节点负责的标识符空间的范围 R

1. Data: $acount \leftarrow 0, h \leftarrow 0, \mathbb{L}[0] \leftarrow 0, \bar{p} \leftarrow \lfloor |\mathbb{I}|/|N| \rfloor$
2. FOR each $\mathcal{I}_{i,j} \in \mathbb{I}$ do
3. SORT e_r in $\mathcal{I}_{i,j}$ orderby $e_r.k$ ASC //按照键值升序排列
4. FOR each $e_r \in \mathcal{I}_{i,j}$ do
5. $acount++$
6. IF $acount \bmod \bar{p}=0$
7. $\mathbb{L}[h++] \leftarrow F(e_r.k)$ //获得每个节点的起始值
8. ENDFOR
9. ENDFOR
10. FOR j from 0 to $|N|-1$
11. $R[j] \leftarrow [\mathbb{L}[j], \mathbb{L}[j+1])$//节点 n_j 负责的空间范围
12. ENDFOR
13. RETURN R

算法 1 输入索引集 $\mathbb{I}$,返回每个局部节点负责的标识符空间的范围 R. 对于局部节点 n_i,其所负责的标识符空间为 $R[i]=[\mathbb{L}[i], \mathbb{L}[i+1])$,所有映射到 $R[i]$ 的索引条目都由 n_i 负责建立局部索引. 例如,共有索引条目 100 条及 10 个节点,则每个节点分配的索引条目数量为 10,那么节点 n_1 所负责的标识符空间 $R[1]=[F(e_{10}.k), F(e_{19}.k))$.

算法 1 具有索引条目聚集分布的特点,即可将一个租户的索引条目分布到尽可能少的局部节点中,从而可减少查询时数据的传输成本. 在一般情况下,一个租户的索引条目数量要小于局部节点容量 $\bar{p}$,所以一个租户的索引条目会分配到一个或两个局部节点. 当数据对象的索引条目数量大于局部节点容量时,分布到局部节点的数量仍能接近理想最佳节点数量,用 OPT(optimal solution)表示该数量,若每个局部节点管理的索引条目数量为 $\bar{p}$,则 $OPT=\lceil |\mathbb{k}_i|/\bar{p} \rceil$. 定理 2 给出了算法 1 分配局部节点数目的限制.

定理 2. 算法 1 保证任意租户 tn_i 的索引条目分布到局部节点的数量 N_{A1} 小于最优值加 1,即 $N_{A1} \leq OPT+1$.

证明. 对于租户 tn_i 的 $\mathbb{k}_i$,设 $\mathbb{k}_i$ 索引条目数量为 $|\mathbb{k}_i|$. 因函数(1)保证属于一个租户的索引条目在标识符的空间分布是连续的,所以算法 1 在分配同一租户的索引条目时,会连续分配到若干个局部节点. 设 n_a、n_b 分别为 $\mathbb{k}_i$ 的索引条目分配到的第一个局部节点与最后一个局部节点,设 N_a^e、N_b^e 为 n_a、n_b 拥有 $\mathbb{k}_i$ 索引条目的数量,那么 $N_a^e \leq \bar{p}$ 且 $N_b^e \leq \bar{p}$. 设 n_f 为 $\mathbb{k}_i$ 索引条目分配到的一个中间局部节点,即 $f \in (a,b)$. 那么 n_f 只会分配到 $\mathbb{k}_i$ 的索引条目,即 $N_f^e=\bar{p}$. 设 N_f 为中间局部节点 n_f 的数量,则 $N_f=N_{A1}-2$. 可得

$$N_a^e+N_b^e \geq |\mathbb{k}_i| \bmod \bar{p} \Rightarrow N_f=\frac{|\mathbb{k}_i|-(N_a^e+N_b^e)}{\bar{p}} \leq \frac{|\mathbb{k}_i|-|\mathbb{k}_i| \bmod \bar{p}}{\bar{p}}=\left\lfloor \frac{|\mathbb{k}_i|}{\bar{p}} \right\rfloor \Rightarrow N_{A1}=N_f+2 \leq \left\lfloor \frac{|\mathbb{k}_i|}{\bar{p}} \right\rfloor+2 \leq OPT+1.$$

证毕.

在应用时,分配算法还可根据实际需要做一些调整. 首先,算法 1 假设各个局部节点的存储能力是同构的. 但在云计算环境中,一般局部节点能力是异构的. 算法 1 中可对变量 $\bar{p}$ 进行改动,使分配的索引条目匹配每个局部节点的存储能力. 其次,算法 1 假设索引条目使用频率是相同的,因此以索引条目的数量作为参考值. 当索引条目的使用频率不同时,可给每个索引条目加上访问频率的权重.

算法 1 第 6 步使分配到每个局部节点的索引条目都是均值 $\bar{p}$,因此保证了均衡化. 算法 1 在多租户索引机制初始化使用. 当索引实例运行后,可利用分裂与合并局部节点的方法[31-32]对各局部节点的索引条目的数目进行动态调整.

对于各局部节点路由表的维护方法,MIMC 仍然使用 Chord 协议[9]维护. 至此各局部节点得到了自己所负责管理的索引条目,并组成了一个 Chord 网络. 下一小节将介绍各局部节点的本地索引管理.

4.3 局部节点的索引组织

本小节给出局部节点的索引组织策略,策略保证索引与相关数据存储在相同节点并利用 B+树组织索引条目. 在上一小节中,通过算法 1 可得到每个局部节点所负责的标识符空间范围,因此局部节点需组织管理映射到其所负责范围的索引条目. 局部节点的索引组织策略分为以下两步完成:

首先,将局部节点 n_h 负责索引条目所涉及的数据对象集合存储到该 n_h. 对于 n_h 所负责的每一个索引条目 e_r,获取 e_r 所属的数据对象集合 D_k,并在 n_h 存储 D_k. 不同租户的数据对象集合采用共享模式的方式存储到宽表中[1]或者文档数据库[33].

其次，令每个索引条目的 v 指向涉及数据对象的物理存储地址，并将索引条目加入到 B+树进行存储. 在局部节点上，本文利用一个 B+树管理所负责的所有索引条目. 局部节点先将每个索引条目变换为$\langle k^s, v\rangle$，再对索引条目建立关于 k^s 的 B+树索引.

局部节点建立索引组织的具体步骤在算法 2 给出. 设$\boldsymbol{E}_i$为局部节点 n_i 所应负责管理的索引条目集合，即$\boldsymbol{E}_i=\{e_r \mid e_r.k^s \in R_i\}$.

算法 2.　局部节点建立索引.

输入：n_i 所应负责管理的索引条目集合$\boldsymbol{E}_i$

输出：B+树索引

1. For each $e_r \in \boldsymbol{E}_i$
2. $D_i \leftarrow$ getDataObject(e_r)
3. IF D_i 尚未存储在本局部节点
4. 在本局部节点存储 D_i
5. ENDIF
6. $k^s \leftarrow$ Function(1)($e_r.k$)
7. $e_r.v$ 指向数据对象的物理节点
8. 以 k^s 为键值将 e_r 插入 B+树
9. ENDFOR

不同于为每个租户分别建立索引实例[12]的情况，算法 2 只维护一个 B+树即可实现对多个数据对象索引的管理，从而节省了缓存资源，降低了管理的复杂度. 并且算法 2 将索引条目与其指向的数据存储到相同的局部节点，得到索引条目即可获得所需数据，提高了查询速度.

算法 2 可减少多租户数据存储的冗余信息. 文献[1,4,6]提出的多租户数据存储模式中每个元组都需要存储租户 ID 与数据对象集合 ID 以标识该数据所属租户及关系，以对元组的所属租户加以区分. 而算法 2 可在租户数据对象集合的主关键字上建立索引，即主键索引，利用主键索引即可提供租户数据定位及全表扫描功能，使得租户数据存储中不再需要存储租户 ID 及数据对象集合 ID，从而减少了多租户数据存储的冗余信息.

本小节通过对索引条目在对等节点上的组织，建立起一个多租户索引机制. 该机制允许在任意局部节点发起租户查询请求 Q，Q 的查询步骤如下：(1)局部节点解析 Q 得到所需查询数据的 k，并利用函数(1)得到 k^s；(2)通过路由表得到 k^s 对应的目标节点，并将 Q 发送到目标节点；(3)通过 Chord 路由协议，Q 最终会路由到租户数据所在节点，并在数据所在节点进行租户数据的查询处理.

5　多租户索引机制的动态管理

在多租户索引机制动态运行过程中，租户数量与租户索引是不断增长的. 平台中随时都有新的租户加入或者新的索引建立，因此需要为新加入的租户或索引动态地分配标识符空间. 如 4.1 节所述，在索引标识符空间映射初始化时，会预留若干租户区域(最大为 Max_{TN})，每个租户也预留了若干数据对象的索引区域(最大为 Max_{IN})，以备索引的新增. 然而，当多租户数据库新增租户 tn_i 的 $i>Max_{TN}$ 时，则没有新的租户区域可供租户使用；当租户 tn_i 新建数据对象索引 $\mathcal{I}_{i,j}$ 且 $j>Max_{IN}$ 时，则没有新的数据对象索引区域可供数据对象索引使用. 针对此问题，可以通过在系统初始化时预留足够数量的租户区域或者数据对象索引区域的方法来解决，然而预留的空间过大会增加 B+树的存储空间. 本文采用对标识符空间进行逐步扩展的方法解决此问题，扩大后的标识符空间可成倍增加多租户数据库可管理的租户数量及数据对象的索引数量.

对标识符空间 S 进行扩展，应尽可能地减少对原有索引机制使用的影响. 首先，不能影响原有索引映射的位置性与隔离性. 其次，索引条目已经分配到局部节点，新形成的标识符空间在局部节点中的分配，不应使局部节点所负责的索引条目有过大的变动，导致大量索引及相关数据迁移. 再次，在动态的标识符空间扩展中不应影响运行中的索引功能，通过维护局部节点路由表保证 Chord 路由寻址的正常运行. 根据以上要求，5.1 节与 5.2 节分别给出新增租户及新增数据对象索引 S 的扩展方法.

5.1　新增租户的标识符空间管理

采用动态扩展标识符空间的方法来处理租户区域分配完毕的问题. 当前标识符空间可管理租户数量为 Max_{TN}，当新分配租户 tn_i 的序号 $i>Max_{TN}$ 时，触发索引机制的标识符空间扩展指令. 扩展标识符空间的方法是将原有标识符空间的空间增加一倍，即$|S'|=2|S|$，如图 3. 因此，索引机制可管理的租户数量为 $Max_{TN}=2Max_{TN}$. 新增数据对象索引的索引条目仍然可用函数(1)在标识符空间 S' 中映射. 算法 3 给出租户区域分配完毕时新增租户的具体算法.

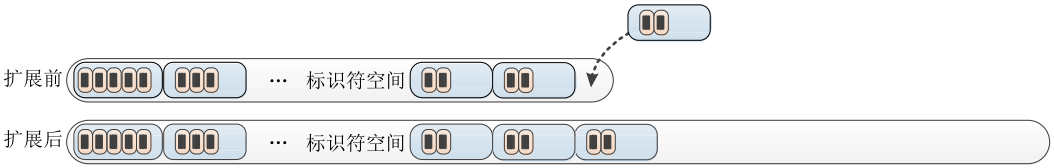

图 3 新增租户的标识符空间管理

算法 3. 租户区域分配完毕时新增租户对 S 扩展.

输入:当前租户数量 $currentNum$,Max_{TN},$|S|$

输出:Max_{TN},$|S|$,租户标识 i,tn_i 的租户区域

1. $i \leftarrow currentNum+1$ //获得租户标识
2. IF $i > Max_{TN}$
3. 寻找节点 n_h,该 n_h 负责标识符 $|S|-1$,即 $|S|-1 \in R_h$
4. n_h 的 $R_h \leftarrow R_h + [|S|, 2|S|)$
5. $|S'| \leftarrow 2|S|$
6. $Max_{TN} \leftarrow 2Max_{TN}$
7. ENDIF
8. 分配 $[|S_{TN}|(i-1), |S_{TN}| \times i)$ 给租户 tn_i 使用
9. $currentNum++$;
10. FOR each $n_i \in N$ //N 为节点集合
11. 更新 n_i 的路由路由条目,使其第 j 个路由条目为 $\langle (l_a + 2^{j-1}) \bmod |S'|, n_x \rangle$
12. ENDFOR

算法 3 对 MIMC 扩展后,对索引机制的使用影响较小.使用函数(1)保持了索引条目映射的位置性与隔离性.新增的标识符空间将分配到对等网络中第 3 步求得的局部节点 n_h 中,其他局部节点负责的标识符空间范围没有变化.若后续增加的租户及数据对象索引引起此局部节点的索引条目数目过多,可将此节点分裂从而动态地平衡节点的数目.标识符空间扩展后,为保证各局部节点仍能正确寻址,需要对各局部节点的路由表进行修正.根据 Chord 路由协议[9]更新每个局部节点的 Chord 路由表.

例如,若当前 $|S|=2^3$,$Max_{TN}=2$,$|S_{TN}|=2^2$,已有租户 2 个,则当新增租户为 tn_3 时,需要扩展 S.那么扩展后 $|S|=2^3\times2=2^4$,$Max_{TN}=4$,tn_3 分配的空间为 $[2^2\times2, 2^2\times3)$.扩展的空间由最后一个节点负责,如图 4,N6 原负责标识符空间[6,7],扩展后负责标识符空间[6,15].路由表的更新如 N2,其路由表增加了一项.

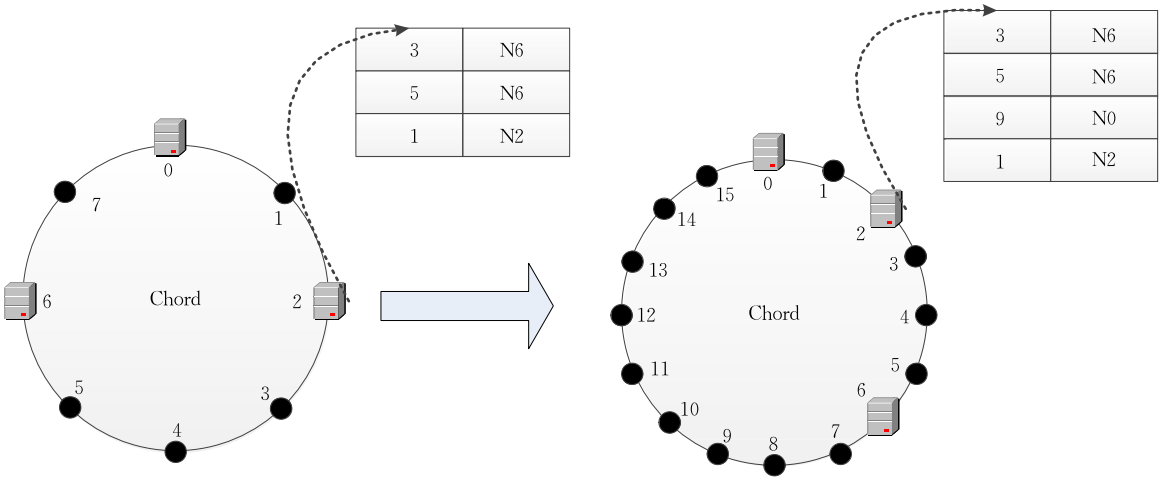

图 4 更新局部节点 Chord 路由表

5.2 新增数据对象索引的标识符空间管理

本小节给出对新增数据对象索引标识符空间管理的方法.一个租户区域在初始时固定大小为 $|S_{TN}|$,多租户数据库根据租户需要后续地新建数据对象索引,当租户 tn_i 新建数据对象索引为 $\mathcal{I}_{i,j}$,$j > Max_{IN}$ 时,需扩展 $|S_{TN}|$.

新建立的数据对象索引应与原有该租户数据对象索引相邻近.所以与新增租户标识符空间管理的直接扩展 S 方法不同,此处采用扩展租户区域 $|S_{TN}|$ 的方法,如图 5,将每个租户区域扩大一倍.以下给出扩展租户区域的主要步骤:

(1) 每个租户区域扩大 1 倍,即 $|S'_{TN}|=2|S_{TN}|$.

(2) 标识符空间也相应扩大一倍,即 $|S'|=2|S|$.

(3) 由于租户区域扩大,对应索引条目的映射

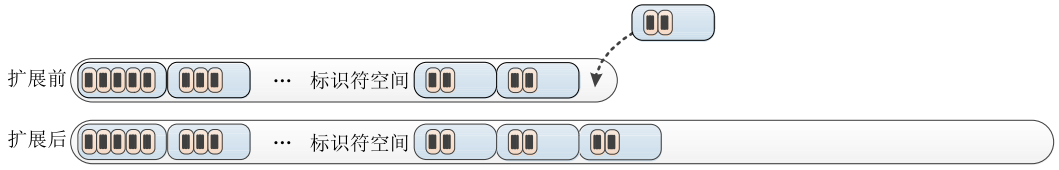

图 5　新增数据对象索引的标识符空间管理

函数也需相应更改. 变更为

$$F'(k)=|S'_{TN}|\times(i-1)+|S_{IN}|\times(j-1)+|S_{IN}|\times(k-min_{i,j})/(max_{i,j}+1-min_{i,j}) \quad (2)$$

根据函数(2)可看出,标识符空间的改变及映射函数的改变并没有影响到索引条目映射的位置性与隔离性.

(4)局部节点采用多 B+树的方法对所负责的索引条目进行管理. 由于现有 B+树中每个节点的 k 值是由函数(1)计算的,而映射函数变更后,多租户数据库则利用函数(2)查找索引条目,导致无法得到正确的目标索引条目. 直观的解决方法是利用函数(2)重建 B+树,但对于实时查询来说重建 B+树的时间成本是不能忍受的. 因此,本文采用多 B+树的方法管理局部节点所负责的索引条目,映射到扩展前租户区域的索引条目仍然利用函数(1)组织与查找 B+树,映射到扩展后租户区域的索引条目则利用函数(2)组织成一棵新的 B+树,由此避免了现有 B+树的重建. 详细方法如下:

每扩展租户区域一次新建一棵 B+树. 令初始 B+树标识号 b 为 0,后续新建的 B+树依次顺序编号;同时对映射函数也进行初始为 0 的顺序编号. 对于新添加的属于 $\mathcal{I}_{i,j}$ 的 e_r,通过函数(3)计算存储 e_r 的 B+树标识号 b,

$$b=\begin{cases}0, & j=1\\ \lfloor \log_{Max^0_{IN}} j-1\rfloor, & j>1\end{cases} \quad (3)$$

其中 Max^0_{IN} 是指系统初始化时为一个租户可建数据对象索引数量的最大值. 那么 e_r 应存储到标识为 b 的 B+树,e_r 的 k^s 值也由映射函数 b 计算.

(5)更改局部节点负责的标识符空间. 对于任意局部节点所负责的标识符空间范围 $[l_a,l_b)$,更新后的所负责的标识符空间范围为 $[l'_a,l'_b)$. l'_a 的计算用函数(3),l'_b 同理.

$$l'_a=\lfloor l_a/|S_{TN}|\rfloor\times|S'_{TN}|+l_a \bmod |S_{TN}| \quad (4)$$

(6)更新每个节点的路由表. 每个路由条目的键值 x^s,可代入函数(4)进行更新;更新后,有部分的路由条目不满足 Chord 路由表规则,需添加与删除对应的路由条目. 路由表更新的时间复杂度为 $O(\log N)$,N 为局部节点数量. 路由表采用异步更新的方式,方法见 5.3 节,从而使得更新路由表的同时不影响查询的跳转与执行.

例如,若当前 $|S|=2^3$,$Max_{TN}=2$,$|S_{TN}|=2^2$,$Max_{IN}=Max^0_{IN}=2$,$|S_{IN}|=2$,租户 tn_1 已有 2 个数据对象索引,则当 tn_1 新建数据对象索引为 $\mathcal{I}_{1,3}$ 时,需要扩展租户区域. 那么扩展后 $|S|=2^3\times2=2^4$,$|S_{TN}|=2^2\times2=2^3$,$Max_{IN}=4$. 根据函数(3),$\mathcal{I}_{1,3}$ 的索引条目应插入到局部节点标识为 1 的 B+树中.

在第(4)步中,采用多 B+树索引的方法管理,避免了因映射函数的改变而需重建 B+树的操作,具有较高的扩展效率. 属于一个 $\mathcal{I}_{i,j}$ 的 e_r 都存储在同一个 B+树中,因此租户查询中的比较操作不受影响. 在处理租户查询时,若需利用数据对象索引 $\mathcal{I}_{i,j}$ 处理数据,则利用函数(3)选择标识为 b 的 B+树进行查询;因 b 号 B+树是利用标识为 b 的映射函数得到的 k^s 建立的,在查询数据值为 k 的数据对象时,需先使用 b 号映射函数计算 k^s,再在该 B+树中进行查找. 因 B+树的选择与 k^s 的计算都是一次计算,所以查询性能不受影响.

标识符空间的改变及映射函数的改变需保证每个局部节点所负责的索引条目应相对稳定,不会导致大量的数据迁移. 在第(5)步中,局部节点所负责的标识符空间范围的维护方法可保证每个局部节点负责的索引条目不变,定理 3 形式化描述了该问题.

定理 3.　$\forall e_r:\langle k,v\rangle$,令 $l_e=F(k)$,$l'_e=F'(k)$,若 $l_e\in[l_a,l_b)$,则 $l'_e\in[l'_a,l'_b)$.

证明.　先证:$l_a\le l_e\Rightarrow l'_a\le l'_e$.

(1)当 $l_a \bmod |S_{TN}|\le l_e \bmod |S_{TN}|$ 时,$l_a\le l_e\Rightarrow\lfloor l_a/|S_{TN}|\rfloor\times|S'_{TN}|\le\lfloor l_e/|S_{TN}|\rfloor\times|S'_{TN}|$,所以 $l'_a\le l'_e$.

(2)当 $l_a \bmod |S_{TN}|>l_e \bmod |S_{TN}|$ 时,若 $\lfloor l_a/|S_{TN}|\rfloor+1>\lfloor l_e/|S_{TN}|\rfloor$,由于左右式均为整数,可得 $\lfloor l_a/|S_{TN}|\rfloor\ge\left\lfloor\frac{l_e}{|S_{TN}|}\right\rfloor\Rightarrow\left\lfloor\frac{l_a}{|S_{TN}|}\right\rfloor+l_a \bmod |S_{TN}|>\left\lfloor\frac{l_e}{|S_{TN}|}\right\rfloor+l_e \bmod |S_{TN}|\Rightarrow l_a>l_e$,与题设矛盾,所以 $\lfloor l_a/|S_{TN}|\rfloor+1\le\left\lfloor\frac{l_e}{|S_{TN}|}\right\rfloor\Rightarrow l'_e\ge\left\lfloor\frac{l_e}{|S_{TN}|}\right\rfloor\times$

$$|S'_{TN}| \geq \left\lfloor \frac{l_a}{|S_{TN}|} \right\rfloor \times |S'_{TN}| + |S'_{TN}| \geq \left\lfloor \frac{l_a}{|S_{TN}|} \right\rfloor \times$$

$$|S'_{TN}| + |S_{TN}| > \left\lfloor \frac{l_a}{|S_{TN}|} \right\rfloor \times |S'_{TN}| + l_a \bmod |S_{TN}| = l'_a.$$

同理,可证 $l_e < l_b \Rightarrow l'_e < l'_b$. 定理 3 得证. 证毕.

5.3 标识符空间扩展的动态路由寻址方法

租户索引机制是动态运行的,在标识符空间扩展时不可能同时更新所有的局部节点. 设 n_e 是已经进行标识符空间扩展的局部节点,n_o 是还未进行标识符空间扩展的局部节点,当租户查询请求 Q 由 n_e 节点路由到 n_o 节点或者由 n_o 节点路由到 n_e 节点时,会出现使用路由表版本的错误. 例如,n_e 节点利用标识符空间扩展后的映射函数 F'_d,计算出 k^s,将索引查询路由到 n_o 节点后,其路由表还未按照函数(3)进行更新,将会路由到错误的节点.

给出的解决方案如下:Q 路由到一个局部节点后,检查标识符空间版本,如果不同,利用该局部节点当前使用的映射函数重新计算一遍 k^s,按照该局部节点的路由表路由即可. 因为根据定理 3 可知一个局部节点负责的索引条目与数据项在标识符空间扩展后是不会发生变化的,所以只要使用相同版本的映射函数与路由表总可以正确指向索引条目所在的局部节点.

6 实验结果与分析

本小节实现了 MIMC 的原型系统,并介绍了 MIMC 与比较对象的实验设置,其比较对象为基于 MongoDB[33] 集群的集中式多租户索引. 考察了查询时间及吞吐量的表现,考察了索引扩展性、隔离性对查询的影响,并测试 5.3 节租户区域扩展的时间成本.

6.1 实验准备

(1) 实验环境

利用 OpenStack 平台建立 8 到 64 个虚拟节点,每个节点安装 64 位 Ubuntu 操作系统,分配 4 个处理器、8 GB 内存、200 GB 磁盘.

实验将 MIMC 与基于 MongoDB 集群的多租户数据库进行比较,以下分别介绍两者的实验设置:

(2) MIMC 实验设置

① 每个节点安装 MongoDB 作为局部节点的数据存储模块,每个节点将所有该节点负责的租户数据都存储到一个 Collection 上;② 利用 MongoDB 索引功能实现本文的局部索引,方法是通过在每条记录上加一项 k^s 的值属性,并建立 B+ 树索引;③ 利用 Java 语言开发对 MIMC 编程实现,主要由 Chord 路由协议模块、Query 网络传递协议模块、MongoDB 本地查询模块、查询测试模块组成;④ 查询具体流程是先给定一个租户的查询,通过 $F(k)$ 值的计算与路由表的查询,得到要处理该 Query 的节点,如果是本地负责查询则交由 MongoDB 处理,如果是其他节点负责,则通过 Chord 协议发送出去.

(3) 基于 MongoDB 集群的集中式多租户索引的实验设置

① 建立 MongoDB 集群;② 在 MongoDB 集群存储多租户数据. 所有租户的数据以共享模式的方式存储在一个 Collection,每个数据对象加上租户标识与数据对象集合标识,并利用数据分片技术[33] 将所有数据对象分布存储到节点中;③ 在 MongoDB 集群中建立索引:首先在集群路由节点建立全局索引,全局索引条目为 $\langle tn_i, D_k, n_x \rangle$,即全局索引记录租户的数据对象集合放置的节点,其次在各分片节点建立 B+ 树结构的局部索引,为提高获得租户数据的效率,建立由租户标识、数据对象集合标识、查询目标属性值组成的联合索引;④ 在查询时,首先对租户查询进行查询转换[34],其次将租户查询提交到集群入口节点,集群入口节点负责从各分片节点获取租户数据并执行查询处理.

多租户数据处理具有在线性、交易性的特点,如 Salesforce 的客户关系管理产品,由于 TPC-C 是模拟一个批发商的货物管理环境的在线事务处理的基准项目,可以部署在多租户数据平台,贴近多租户数据的应用场景,因此本文采用 TPC-C[35] 的业务模型模拟生成租户数据. 实验模拟租户对基准业务模型的数据模式定制,对于每个租户,实验中以原始数据表为基准,随机增加 1 到 5 个属性列;同时加上租户标识列,用以区分每条记录所属的租户. 在实验中,主要为租户生成定单表与定单分录表数据,并为定单表的 O_ID、定单分录表的 OL_O_ID 建立索引.

为衡量 MIMC 在线事务处理系统的性能及其可伸缩性,本实验考察 3 种查询:点查询、范围查询和连接查询. 点查询是租户对一个数据对象集合发起的等值查询,在实验中针对定单表(ORDER)查询的定单号属性进行条件查询,查询谓语为

POINTQUERY: SELECT * FROM ORDER WHERE O_ID=γ

范围查询是指租户对一个数据对象集合的属性给出最大值、最小值从而获得一个结果集,在实验中针对定单表(ORDER)的定单号进行范围查询,从而得到运单号结果集,查询谓语为

RANGEQUERY：SELECT O_CARRIER_ID FROM ORDER

WHERE O_ID＞α AND O_ID＜β

对于连接查询，本实验只考察两个数据对象集合的等值连接查询，对定单表与定单分录表(ORDER_LINE)连接取相关信息. 由于 MongoDB 不支持连接操作，实验在平台利用哈希连接的方法完成连接查询任务. 连接查询谓语为

JOINQUERY：SELECT ORDER.O_CARRIER_ID, ORDER_LINE.OL_SUPPLY_W_ID, ORDER_LINE.OL_QUANTITY FROM ORDER, ORDER_LINE WHERE O_ID＞α AND O_ID＜β AND O_ID＝OL_O_ID

为满足大多数数据类型的索引对哈希空间位数的需求，可设置$|S_{IN}|$为2^{128}，该S_{IN}最大可保证 128 位长整型及 16 位字符型在映射中无冲突. 在初始时，设置可管理租户数量为$Max_{TN}=2^{16}$，每个租户可建立$Max_{IN}=2^8$个索引，那么$|S|=2^{128}\times 2^{16}\times 2^8=2^{152}$，即标识符空间的位数为 152. 在实验中可利用 java. math. BigInteger工具进行大整型运算. Max_{TN}与Max_{IN}的初始设置是一个较小的值，若系统管理租户与索引数量超过两值，可通过第 5 节空间扩展方法解决. 空间扩展会导致 S 的位数的增加，本文通过实验测试 S 的位数在 64 到 256 位变化时，对平均查询时间的影响小于 1 ms.

6.2　MIMC 的查询时间

本小节对 MIMC 的查询时间进行测试. 首先以租户数量作为变量，数量从 160～6400 个变化，考察使用 MIMC 的查询时间. 建立 32 个局部节点管理租户数据，每个租户的数据量大小为 100 000 条，每个租户同时发出 3 个查询请求. 基于 MongoDB 集群的集中式多租户索引作为比较对象，其 3 种查询分别称为集中索引点查询、集中索引范围查询、集中索引连接查询. 由于集中索引连接效率过低，实验中不再考察.

实验结果如图 6，MIMC 的点查询、范围查询、连接查询有着较好的表现. 随着租户数量的增多，MIMC 的查询时间优势更加明显，因为集中索引使得入口节点的并发查询过多，从而使得查询等待执行的时间较长. 在租户数量等于 160 时，MIMC 范围查询仍然比集中索引范围查询消耗的时间少，因为范围查询结果集较大，MIMC 直接将结果传输给查询发起节点从而减少了网络传输时间.

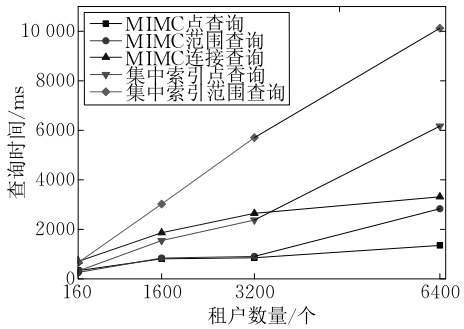

图 6　以租户数量比较查询时间

其次考察每个租户数据量作为变量，数据量从 1000～1 000 000 条记录变化，32 个节点，设定 256 个租户分布在这些节点上，每个租户同时发起 5 个查询请求. 实验结果如图 7，MIMC 点查询的时间消耗不受数据量大小的影响，而 MIMC 范围查询与连接查询随着数据量的增加对节点内存要求增多，查询时间平缓增长. 集中索引点查询与集中索引范围查询因为入口节点的并发查询队列等待时间过长，所以查询时间长于 MIMC.

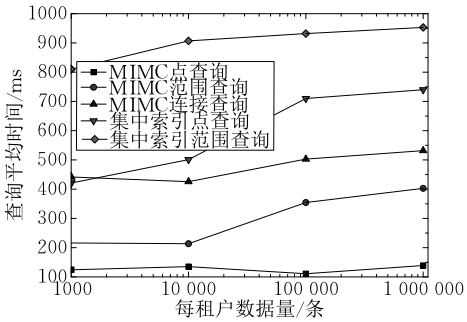

图 7　以数据量大小比较查询时间

6.3　MIMC 的扩展性

本小节考察 MIMC 的扩展性能. 首先测试节点数量的变化对查询吞吐量的影响. 节点数量范围为 8～64，同时每增加一个节点对应增加 10 个租户，每个租户的数据对象数量为 100 000 条. 实现结果如图 8 所示，MIMC 的 3 种查询的吞吐量与节点数量成线性关系，特别是点查询与范围查询，而连接查询

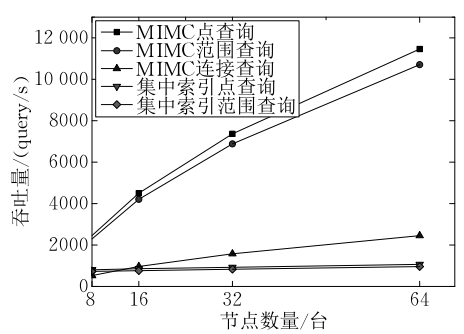

图 8　节点数量对吞吐量的影响

消耗的 CPU、内存资源较多，所以吞吐量增长较慢. 由于入口节点的瓶颈，集中索引点查询与范围查询的吞吐量并没有随着节点数量的增加而增长.

其次测试节点数量的扩展对查询时间的影响. 分别测试 MIMC 点查询时间与集中索引点查询时间，*5 是指每增一个节点对应增加 5 个租户，*50 是指每增一个节点对应增加 50 个租户，且设定每个租户同时发起 5 个查询，每个租户的数据量为 100 000 条. 实验结果如图 9，可观察到 MIMC 的点查询时间并不随着节点规模的扩大而显著延长，这是由于租户查询在节点规模增大时，只是指数级地增加路由的跳数，而每一跳消耗的时间数量级为 10 ms. 对于集中索引点查询，查询时间随着租户与节点规模的扩大而增长较多，特别是在 *50 时过多的时间消耗在队列等待中.

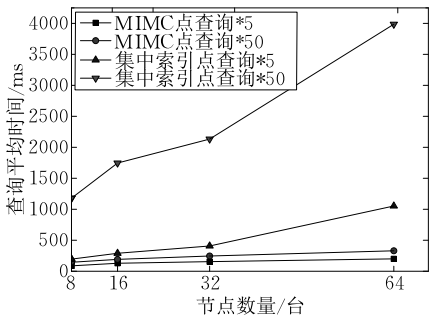

图 9 节点数量对查询时间的影响

以上实验表明 MIMC 可以通过增加节点数量适应大规模多租户数据查询处理的需求. 由图 6、图 7 可看出，当节点数量一定时，影响查询效率的主要因素是租户数量，而每个租户的数据量影响较小. 这是因为同时发出的查询请求数量与租户数量成比例，当查询请求数量较多时，查询任务会在队列等待中消耗过多的时间. 根据 P2P 结构的特性，MIMC 会将查询请求分散到各节点中，从而避免各节点出现性能瓶颈. 图 6 结果表明，在当前实验环境下，当租户与节点数量比小于 100 时，点查询与范围查询的时间成本较低. 图 9 表明，同时增加租户数量与节点数量，查询时间增长缓慢，这是因为增加的跳数是节点数量的对数级. 因此，使用 MIMC 时，可根据性能需要设置好租户与节点的比例关系，当租户数量增加时，按比例增加节点数量即可.

6.4 租户区域扩展的时间成本及查询影响

本小节考察 5.2 节扩展租户区域的时间成本及扩展后对查询的影响. 租户区域扩展时，主要耗时在步骤(4)B+树的维护及步骤(6)路由表的维护上. 在步骤(4)中，针对 B+树失效的问题，比较直观的解决方法是重建 B+树，本文还提出了一种多 B+树的方法. 实验将两种方法进行比较，测试租户区域扩展的时间. 实验还利用点查询测试两种方法建立索引结构对查询的影响.

实验在一个局部节点上进行，管理 8 个租户，每个租户的数据对象数量从 1000～1 000 000 条变化. 对于租户区域扩展的时间成本，如图 10 的柱状图比较，多 B+树方法扩展时只需新建一个空 B+树及维护路由表，与重建 B+树方法相比有着明显优势，特别是在数据对象数量较大时. 对于查询性能，如图 10 的折线比较，以上两种方法平均查询时间相似，这是由于多 B+树方法只比重建 B+树方法多一步 B+树的选择，通过一次计算即可完成.

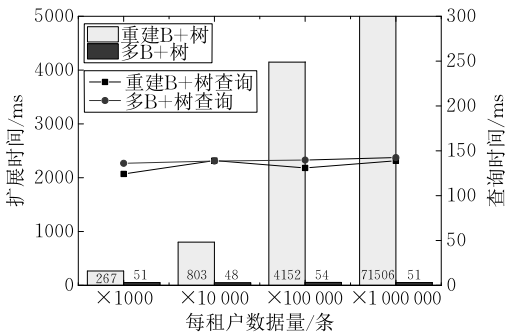

图 10 租户区域扩展的时间及查询时间

6.5 隔离性对查询的影响

本小节考察隔离性对租户查询的影响. 4.1 节证明了 MIMC 的映射函数具有隔离性，租户使用索引查询时只检索到自己的数据，由此提高了查询效率. 为验证隔离性对租户查询性能的提升，本节将 MIMC 与 DBMS-like[19] 进行比较.

DBMS-like 实验设置：DBMS-like 是 Chen 等人[19]提出的一种针对 DaaS 的云中关系运算索引机制，其将索引保序地发布到 P2P 网络. 在实验中，所有租户的数据对象都使用其保序函数在云中映射，并采用 Chord 协议组织 P2P 网络. 由于相同 k 值但不同租户的索引条目会映射到 S 中的相同位置，所以此方法不能保证租户间的隔离性.

基于 MongoDB 集群的集中式多租户索引也不具备租户隔离性，但其因资源争用会影响查询时间，所以不再考察.

实验中，多租户数据放置于 8 个节点，租户数量由 32 到 256 变化，每个租户数据量为 100 000. 为避免资源争用对查询时间的影响，实验测试只有一个租户发起查询的时间成本，测试的查询类型为点查询与范围查询. 实验结果如图 11，MIMC 范围查询比

DBMS-like 范围查询耗时少，这是由于 DBMS-like 检索到的数据含有大量其他租户的数据，需在这些数据中做进一步的筛选. 而且当租户数量增多时，DBMS-like 范围查询的耗时急剧增加，而 MIMC 范围查询的耗时基本不变，这是因为 MIMC 的租户索引是隔离的，不受租户数量的影响. 而对于 MIMC 点查询与 DBMS-like 点查询，租户的隔离性对其查询耗时没有影响，这是由于 DBMS-like 方法只等值的检索一条或几条数据，并不需处理其他大量的无关数据. 因此，租户隔离性对范围查询有较大影响.

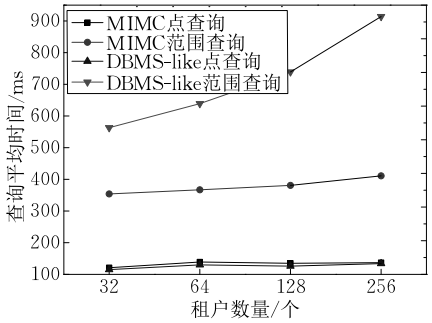

图 11　租户隔离性对查询时间的影响

6.6　MIMC 的复杂查询性能

本小节测试 MIMC 在较大数据量下复杂查询的性能. 实验采用 TPC-H 的数据模式生成租户数据，并在 TPC-H 的数据模式上加入租户标识. 节点数量为 8 个，租户数量由 16 到 64 变化，每个租户数据量为 10 000 000. 实验以 Q6 与 Q17 作为测试对象，Q6 是针对 LINEITEM 的单表统计，Q17 是涉及 LINEITEM 与 PART 的双表数据分析查询. 在实验中，对 P_PARTKEY、L_PARTKEY、L_SHIPDATE 属性建立索引，设置每个租户同时发起 1 个查询. 如图 12，实验表明，利用 MIMC 比基于 MongoDB 集群的集中式多租户索引查询耗时少. 特别是对于占用资源较多的 Q17，因后者在查询时产生资源争用情况，租户数量越多耗时越长. 因此，对于大数据量且较为复杂的租户查询，MIMC 仍有较好的查询性能.

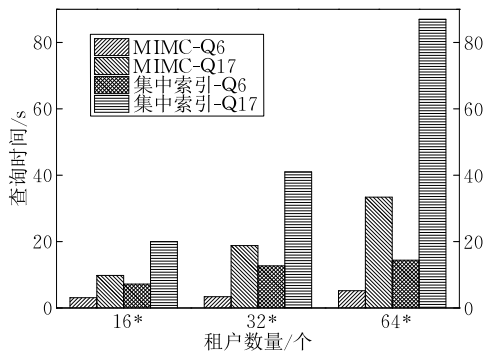

图 12　以租户数量比较复杂查询时间

7　结　论

本文提出了一种通过改进映射方法的 Chord 多租户索引机制，其提供的映射函数及标识符空间分配算法可以隔离、保序、均衡地将租户的索引条目分配到云中节点上. 提出的动态索引管理方法，适应了租户数量及数据无限增长的需求. MIMC 的局部节点是对等网络节点，相对于集中式索引其不存在性能瓶颈. 最后实验表明，当 MIMC 在租户规模较大时，查询时间与吞吐量有较好的性能表现.

参　考　文　献

[1] Weissman C D, Bobrowski S. The design of the force.com multitenant internet application development platform//Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems. Providence, USA, 2009: 889-896

[2] Malawski M, Kuzniar M, Wjcik P, Bubak M. How to use Google App Engine for free computing. IEEE Internet Computing, 2013, 17(1): 50-59

[3] Chong F, Carraro G. Architecture Strategies for Catching the Long Tail. Redmond, WA, USA: Microsoft Corporation, 2006

[4] Aulbach S, Jacobs D. A comparison of flexible schemas for Software as a Service//Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems. Providence, USA, 2009: 881-888

[5] Ni Jiacai, Li Guoliang. Adapt: Adaptive database schema design for multi-tenant applications//Proceedings of the 21st ACM International Conference on Information and Knowledge Management. Maui, USA, 2012: 2199-2203

[6] Aulbach S, Grust T. Multi-tenant databases for software as a service: Schema-mapping techniques//Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. Vancouver, Canada, 2008: 1195-1206

[7] Cryans J-D, April A, Abran A. Criteria to compare cloud computing with current database technology//Dumke R R, Braungarten R, Büren G, et al, eds. Software Process and Product Measurement. New York: Springer, 2008

[8] Oram A. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. CA, USA: O'Reilly Media, Inc., 2001

[9] Stoica I, Morris R. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149-160

[10] Das S, Agrawal D. ElasTraS: An elastic, scalable, and self-managing transactional database for the cloud. ACM Transactions on Database Systems, 2013, 38(1): 5

[11] Jacobs D, Aulbach S. Ruminations on multi-tenant databases //Proceedings of the Business, Technologie und Web. Aachen, Germany, 2007: 514-521

[12] Hui Mei, Jiang Dawei. Supporting database applications as a service// Proceedings of the 25th IEEE International Conference on Data Engineering. Shanghai, China, 2009: 832-843

[13] Zhang Xiangyu, Ai Jing. An efficient multi-dimensional index for cloud data management//Proceedings of the 1st International Workshop on Cloud Data Management. Hong Kong, China, 2009: 17-24

[14] Papadopoulos A, Katsaros D. A-tree: Distributed indexing of multidimensional data for cloud computing environments// Proceedings of the Cloud Computing Technology and Science (CloudCom). Athen, Greece, 2011: 407-414

[15] Wu Sai, Jiang Dawei. Efficient B-tree based indexing for cloud data processing. Proceedings of the VLDB Endowment, 2010, 3(1-2): 1207-1218

[16] Wang Jinbao, Wu Sai. Indexing multi-dimensional data in a cloud system//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indiana, USA, 2010: 591-602

[17] Ding Linlin, Qiao Baiyou. An efficient quad-tree based index structure for cloud data management. Web-Age Information Management. New York, USA: Springer, 2011

[18] Wu Sai, Jiang Dawei. Efficient btree based indexing for cloud data processing. Proceedings of the VLDB Endowment, 2010, 3(1): 1207-1218

[19] Chen Gang, Vo Hoang. A framework for supporting DBMS-like indexes in the cloud. Proceedings of the VLDB Endowment, 2011, 4(11): 702-713

[20] Ratnasamy S, Francis P. A Scalable Content-Addressable Network. New York, USA: Association for Computing Machinery, 2001

[21] Jagadish H V, Ooi B C. BATON: A balanced tree structure for peer-to-peer networks//Proceedings of the 31st International Conference on Very Large Data Bases. Trondheim, Norway, 2005: 661-672

[22] Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems//Proceedings of the Middleware 2001. Heidelberg, Germany, 2001: 329-350

[23] Karger D, Lehman E. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web//Proceedings of the 29th Annual ACM Symposium on Theory of Computing. El Paso, USA, 1997: 654-663

[24] Pearson P K. Fast hashing of variable-length text strings. Communications of the ACM, 1990, 33(6): 677-680

[25] Datar M, Indyk P. Locality-sensitive hashing scheme based on p-stable distributions//Proceedings of the 20th Annual Symposium on Computational Geometry. Brooklyn, USA, 2004: 253-262

[26] Nishimura S, Das S. MD-HBase: A scalable multi-dimensional data infrastructure for location aware services// Proceedings of the the 12th International Conference on Mobile Data Management. Luleå, Sweden, 2011: 7-16

[27] Lawder J K, King P J H. Querying multi-dimensional data indexed using the Hilbert space-filling curve. ACM Sigmod Record, 2001, 30(1): 19-24

[28] Cormen T H, Leiserson C E. Introduction to Algorithms. London, UK: MIT press Cambridge, 2001

[29] Surana S, Godfrey B. Load balancing in dynamic structured peer-to-peer systems. Performance Evaluation, 2006, 63(3): 217-240

[30] Karger D R, Ruhl M. Simple efficient load balancing algorithms for peer-to-peer systems//Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures. Barcelona, Spain, 2004: 36-43

[31] Li Min, Chen Enhong, Sheu Phillip Cy. A chord-based novel mobile peer-to-peer file sharing protocol//Zhou Xiaofang, et al, eds. Frontiers of WWW Research and Development-APWeb 2006. New York, USA: Springer, 2006: 806-811

[32] Novak D, Zezula P. M-Chord: A scalable distributed similarity search structure//Proceedings of the 1st International Conference on Scalable Information Systems. Hong Kong, China, 2006: 19-20

[33] Chodorow K. MongoDB: The Definitive Guide. CA, USA: O'Reilly Media, Inc., 2013

[34] Liao Chun-Feng, Chen Kung. Toward a tenant-aware query rewriting engine for Universal Table schema-mapping// Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science. Taipei, China, 2012: 833-838

[35] Leutenegger S T, Dias D. Modeling study of the TPC-C benchmark//Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. Washington, USA, 1993: 22-31

ZOU Li-Da, born in 1981, Ph. D. candidate. His research interests include cloud data management and query optimization.

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.

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

WANG Zhen-Kun, born in 1991, M. S. candidate. His research interests focus on database.

Background

Multi-tenant database is an important foundation of software as a Service (SaaS), and it becomes a popular issue in this field. One of the greatest characteristics in SaaS is single- instance and multi-tenancy, i. e., more than one tenants use an application. Tenants purchase and use compute resources as needed, which greatly save costs for tenants. Multi-tenant database can simultaneously provide data management and access service for multi-tenants. SaaS application is deployed on top of multi-tenant database and tenants access their own data in multi-tenant database through the application.

Indexing mechanism is an important issue of efficient query in the field of multi-tenant data management. This paper focuses on establishment and maintenance of highly efficient indexing mechanism. Most related works address the problem of index isolation for multi-tenant data in shared schema. They separately establish indexes for tenants and tenants only access their own data through isolated index, which can improve access efficiency. However, massive multi-tenant data should be deployed in cloud computing environment. At present multi-tenant indexing mechanism in clouds has not been discussed yet. The challenging problems to be solved include how to distribute multi-tenant indexes in clouds, how to achieve isolation of multi-tenant indexes in clouds and how to dynamically manage the increasing tenants and indexes.

This paper proposed a Multi-tenant Indexing Mechanism Based on Improved Chord Mapping Approach (MIMC), which can distribute index entries to nodes in clouds isolated, orderly and evenly. The presented approach of dynamic index management can accommodate the scenario of infinitely increasing number of tenants and their data. As the nodes form a P2P network, there is no performance bottleneck for MIMC compared with centralized indexing mechanism. Experiments show that when tenants are on a large scale, query efficiency is 2 to 6 times faster and throughput is improved by 1.5 to 7 times than that of centralized indexing mechanism.

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61572295, 61303085, the Natural Science Foundation of Shandong Province under Grant Nos. ZR2013FQ014, ZR2014FM031, the Science and Technology Development Plan Project of Shandong Province No. 2014GGX101047, and the Shandong Province Independent Innovation Major Special Project Nos. 2015ZDJQ01002, 2015ZDXX0201B03.