Journal of Computer Applications
计算机应用,2016,36(1):1-7,12
ISSN 1001-9081
CODEN JYIIDU
2016-01-10
http://www.joca.cn

文章编号:1001-9081(2016)01-0001-07
DOI:10.11772/j.issn.1001-9081.2016.01.0001

集群环境下分布式索引的实现

翁海星[1],宫学庆[1],朱燕超[1],胡华梁[2*]
(1.华东师范大学 数据科学与工程研究院,上海 200062; 2.浙江理工大学 经济管理学院,杭州 310018)
(*通信作者电子邮箱 jmxyandy@126.com)

摘 要: 针对分布式存储系统上使用非主键访问数据带来的性能问题,探讨在分布式存储系统上实现索引的相关关键技术。在充分分析分布式存储特征的基础上,提出了分布式索引设计和实现的关键点,并结合分布式存储系统的特点及相关的索引技术,讨论了索引的组织形式、索引的维护和数据一致性等问题;然后基于如上的分析,选择在分布式数据库系统 OceanBase 开源版本上,设计和实现分布式索引机制,并通过基准测试工具 YCSB 进行性能测试。实验结果表明,虽然辅助索引会对系统性能产生影响,但因为充分考虑了系统特征及存储特点,在不同数据规模下,该索引都能够将性能影响控制在 5% 以内。另外,使用冗余列的方式,能进一步将该索引的性能提升 100%。

关键词: 分布式存储;分布式索引;辅助索引;索引维护;OceanBase
中图分类号: TP311 **文献标志码:** A

Implementation of distributed index in cluster environment

WENG Haixing[1], GONG Xueqing[1], ZHU Yanchao[1], HU Hualiang[2*]
(1. *Institute for Data Science and Engineering, East China Normal University, Shanghai* 200062, *China*;
2. *College of Economic and Management, Zhejiang Sci-Tech University, Hangzhou Zhejiang* 310018, *China*)

Abstract: For performance issues brought by using non-primary key to access data on a distributed storage system, key technologies were mainly discussed to the implementation of indexing on a distributed storage system. Based on the rich analysis of new distributed storage features, the keys to design and implementation of distributed index were presented. By combining characteristics of distributed storage system and associated indexing technologies, the organization and maintenance of index, data concurrency and other issues were described. Then, the distributed indexing mechanism on the open source version of OceanBase, which is a distributed database system, was designed and implemented. The performance tests were run on the benchmarking tool YCSB. The experimental results show that the distributed auxiliary index will degrade the system performance, but it can be controlled within 5% under different data scale because of the consideration of system features and storage characteristics. In addition, it can increase index performance by even 100% with a redundant colume way.

Key words: distributed storage; distributed index; auxiliary index; index maintenance; OceanBase

0 引言

为了能够响应和处理互联网级的访问负载,越来越多的应用开始使用基于分布式存储的数据库系统。这些系统的一个共同特征是将数据分片后冗余存储在集群中的多个节点上,从而保证系统的扩展性、可靠性和可用性。按记录主键划分数据是数据分片的主要方式。通过主键属性能够快速定位数据所在的分片,而通过非主键属性往往需要对所有数据分片进行扫描。出于性能的考虑,很多分布式存储系统不建议使用非主键的方式访问数据。

当无法避免使用非主键属性来访问数据时,分布式存储系统的性能成为开发人员需要面对的问题。创建辅助索引是数据库系统改善访问性能的重要手段,在传统关系数据库和分布式数据库领域已经有相当成熟的索引技术。在将这些索引技术应用于新的分布式存储系统时,必须充分考虑每个系统的架构特点,设计和实现能够满足应用场景需求的索引结构。这一领域的研究工作也是近年来的一个热点。

本文对分布式存储系统的特点及相关的索引技术进行了分析,探讨了在分布式存储系统上实现辅助索引需要考虑的问题,包括索引的组织形式、索引分布和索引的维护。同时,本文在对分布式数据库系统 OceanBase 的开源版本进行介绍的基础上,根据以上的分析,为该系统设计和实现了辅助索引机制,并且通过基准测试工具 YCSB(Yahoo Cloud Serving Benchmark)进行了性能测试。测试结果表明,虽然增加辅助索引会对系统的性能产生影响,但这种影响是在可接受范围内并且是稳定的;同时,辅助索引能够极大地提高非主键访问的性能。

1 背景和相关工作

1.1 分布式存储系统

分布式存储系统已经广泛应用于工程实践中,例如 Google BigTable[1]、Amazon Dynamo[2]、Windows Azure

收稿日期:2015-09-08;修回日期:2015-10-08。
基金项目:浙江省自然科学基金资助项目(LY12F02044);国家自然科学基金重点项目(U1401256)。
作者简介:翁海星(1993-),男,贵州都匀人,硕士研究生,主要研究方向:分布式数据库、大数据管理; 宫学庆(1974-),男,上海人,教授,博士,主要研究方向:数据库、社交网络分析; 朱燕超(1992-),男,江苏溧阳人,博士研究生,主要研究方向:分布式数据库; 胡华梁(1974-),男,江西武宁人,副教授,博士,主要研究方向:数据管理。

Storage[3]等，很多大型互联网企业都采用成熟的分布式存储系统方案，以实现对大规模数据的存储、计算以及价值信息的提取。同时，围绕HBase[4]、Cassandra[5]等的开源分布式存储系统展开的相关设计、优化和研究成为备受关注的课题。

相比传统的单点存储系统的部署需要高端服务器来提供高性能数据处理和存储支持，分布式存储系统通常由数量众多、成本较低的普通服务器组成，集群中每个服务器节点都会存储并管理系统的部分的数据。分布式系统中，数据一般采用多副本冗余存储，当某个节点出现故障时，该节点上的数据可以在其他节点上找到相应副本，仍然可以被访问。分布式存储系统一般都具有良好的线性可扩展性，通过增加存储节点可以线性增加系统的处理与存储能力。

按记录主键对数据进行划分，将数据分布存储在多个节点，是数据分布的主要手段。大多数分布式存储系统将数据组织成一个分布式Hash表，或者实现分布式平衡树(如B+树)。例如采用基于Hash存储引擎的Amazon Dynamo[2]，可以看作是哈希表的持久化实现，通过一致性哈希机制[6]将数据项以key(主键)、value(内容)为主要形式分片到不同的数据节点，这样的设计实现只能提供基于指定主键的数据库基本操作(Create，Retrieve，Update，Delete，CRUD)功能。而在Google BigTable[1]的实现中，系统以表格为单位组织数据，每个表上的数据按照主键(row key)有序，组成一个B+树，一个叶子节点包含表的一个前开后闭的主键的范围，每个计算节点可以看成存储了分布式B+树的一个或几个叶子节点，支持根据主键的CRUD以及范围查找。

通过维护数据分片以及集群节点的信息，分布式存储系统能够很好地处理客户端的主键查询请求。根据数据分布情况及集群负载，系统会选择合适的存储节点进行访问，并发地向它们发送对应主键(主键范围)的查询请求，最后将从每个节点读取到的本地数据整合后返回给客户端；但是，对于非主键的查询请求，系统无法根据主键确认数据位置，只能对所有节点的数据进行全部扫描，访问性能极受影响。

1.2 分布式索引

创建辅助索引，是提高访问性能的重要手段。传统的单点存储系统，已经拥有很成熟完善的索引机制，但由于系统架构、数据存储的根本改变，单点存储系统的辅助索引技术不能直接运用在分布式存储系统当中。二者的主要区别在于，对于分布式系统，海量数据上的辅助索引极可能拥有与原数据相仿的规模，这就意味着辅助索引也需要分布式存储。在庞大的数据规模下，分布式存储系统的辅助索引管理，组织形式的设计，以及访问和维护的机制，都存在传统单点存储系统未曾有过的新问题。

目前已有许多围绕不同分布式存储系统展开的辅助索引研究工作。对于如Cassandra[5]、Riak[7]等，不依赖中心节点管理集群，拥有环状结构的系统，基于P2P(Peer-to-Peer)的索引实现有益于高可扩展性，LSH(Locality-Sensitive Hashing)[8]是在Chord[9]的基础上实现的分布式索引，通过位置敏感的哈希算法来获得属性值区间标识与存储节点的位置。对于Master/Slave模式的分布式存储系统，也有很多构建辅助索引的研究工作：Hindex(HuaweiIndex)[10]、IHBase(Index_Hbase)[11]实现了对于每个节点数据维护局部索引的方案，这种索引的设计适用于范围扫描查询，不适合随机读取的场景；CG-Index(Cloud Global Index)[12]是基于亚马逊云计算平台(Amazon Elastic Computing Cloud，Amazon EC2)[13]的一个二级索引实现机制，其基本思想是每个节点不仅维护一个局部B+树，还维护全局的CG-Index，通过访问CG-Index可以确定需要在哪些节点上查询局部索引，支持高性能的随机读取；CCIndex(Complemental Clustering Index)[14]将索引看作另一种形式的数据并存成表，使用可靠的互补校验表(Complemental Check Table，CCT)来代替这些表的备份实现了索引容错和恢复，大幅度减少了索引的存储开销；Diff-Index(Different-schemes Index)[15]考虑了不同场景和需求下的索引-数据表的同步机制。

不同架构的分布式存储系统，以及不同的应用场景下，索引的设计和实现都有一定的差异，但在对一个分布式存储系统设计索引时，都应当结合系统特点以及应用场景，围绕关键的索引技术进行设计，下文将分析探讨这些要点。

2 分布式存储的索引组织

2.1 索引组织形式

分布式存储系统的辅助索引，也需要分布式存储，即集群环境中的每一个节点，都可能保存着一部分的索引信息。维护在每个计算节点中的索引可以采用Hash以及B+树等结构组织，也可以将索引以表的形式组织和存储，将索引看作是另一种形式的数据。单个节点中的索引组织形式，在不同的场景下，不同的方案各有其特点和优势。

Hash适用于典型的Key-Value键值存储系统，以及对于随机读取性能要求较高的场景，其原理是对索引属性进行哈希运算。Hash结构的查询响应速度最快，理想情况下哈希索引的插入和查询时间都是$O(1)$，但对于组合索引，在计算Hash值时是将索引键合并后一起计算Hash值，而不是单独计算，所以，当用索引列前缀的形式查找记录时，Hash索引无法起作用；Hash结构的索引也不能支持范围查询。

B+树支持范围查找和前缀查找，能够提升非主键范围查询的性能。B+树的特点是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度，因此广泛应用于多数关系数据库当中。值得注意的是，维护B+树辅助索引需要更高的成本，因为当树的节点饱和后，会导致节点的分裂。在分布式环境下，还需考虑到集群的负载均衡，整个B+树需要不断动态调整，树的叶子节点除了分裂外还可能发生合并。

如前文提到的CCIndex，还有一种形式，是利用分布式存储系统的可靠性和可扩展性，以表的形式将索引存储起来，将索引看作是另一种形式的数据。此时分布式环境下庞大的索引也可以实现负载均衡以及节点故障时的副本切换容错，这适用于对索引的可用性要求较高的应用场景。还可以将频繁查询的属性字段冗余存储在索引里，当需要查询冗余列属性，就不需要再访问数据表，而直接从索引中返回记录字段，减少了一次数据或者网络的访问。

2.2 局部vs全局

对于分布式存储环境下索引的实现，本文按照逻辑结构将其分为两类：局部索引方案和全局索引方案，并分别探讨两种方案的特点和适用场景。

局部索引方案是指将分布式索引局部化到各个节点上，每个节点创建和维护本地数据的索引，不同节点上的索引信

息是相互独立的。这种解决方案的优势在于,数据和索引存储在相同的节点,不需要系统维护索引在集群上的分布信息,简化了索引维护工作,如图 1 所示。

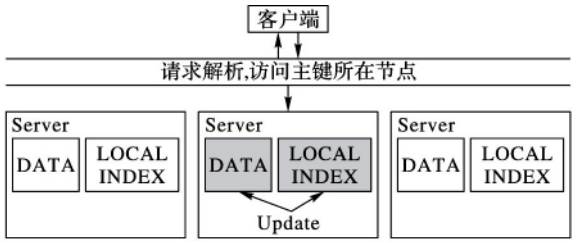

图 1　局部索引数据更新流程

然而,对于选择结果集很小的随机读取请求,该方案仍需要访问所有节点的局部索引,如图 2,增加了一定的网络通信开销和节点负载。对于这种问题,系统可以通过设计和实现额外的索引维护机制来避免访问不必要的节点,但是这样的维护机制也同样增加了系统开销。

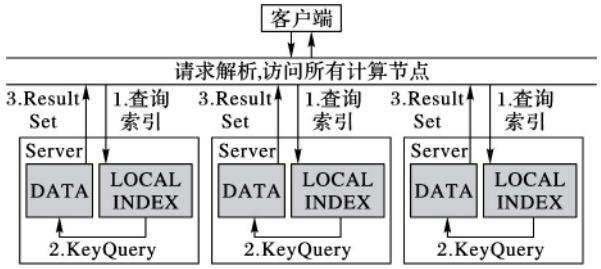

图 2　局部索引方案数据访问流程

在全局索引的方案中,全局的索引根据字段按照规则(如排序)被分片到集群的各个节点,因此一个节点维护的不仅仅是本地数据的索引; 同时也要维护索引在集群的分布信息,这样响应随机读取查询的时候,可以定位到维护相应索引的节点,很快得到对应数据的主键值,如图 3,避免访问没有数据的计算节点,提高了查询性能。此时,索引的维护成本有所增加,因为更新数据的时候,该数据对应的索引很可能在其他的节点上,需要同另外的节点网络通信修改索引,如图 4。

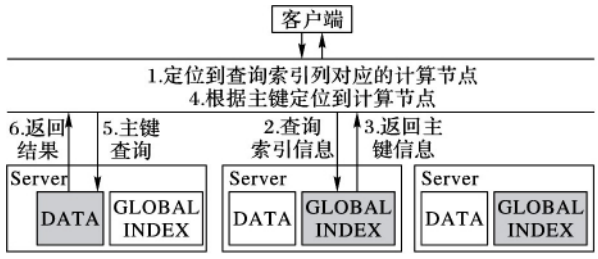

图 3　全局索引方案数据访问流程

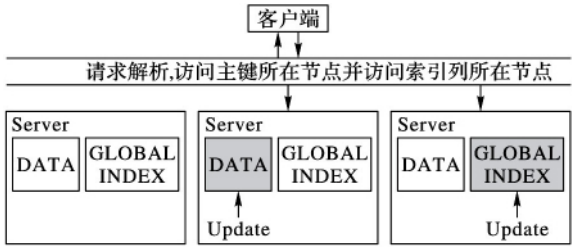

图 4　全局索引方案数据更新流程

在分布式存储系统上实现辅助索引,局部索引方案的索引维护工作可以得到简化; 因为每次查询都要访问所有节点的局部索引,所以适合于范围查询而不适合结果集很小的随机读取场景。全局索引需要维护索引在集群的分布信息,可以直接访问索引所在的节点,能够很好地支持随机读取; 同时,实现全局索引时应当使维护索引更新的网络交互最小化,减少全局索引的维护成本。

2.3　分布式索引的维护

分布式索引生效(可用于提供服务)的时机往往取决于各个节点的局部数据量。对于在空表上新建的索引,只需在更新数据表的同时更新索引,此时新建的索引是可以用于提供服务的,但对于在已有数据的表上创建索引的情形,索引不能立即提供服务。原因在于分布式存储系统中,无论分布式索引采用局部索引还是全局索引,都需要各个节点完成本地索引的构建,否则会导致索引与数据表之间的数据不一致。

采用不同的索引方案,构建本地索引的过程不尽相同。采用局部索引方案构建索引时,每个节点只需负责在本地的数据上对索引字段构建局部索引。而对于全局索引方案,每个节点不仅要处理本地的数据,还可能需要与其他节点通信,获取其数据记录,才能构建出完整且正确的索引。索引创建完成后,在对数据记录更新的同时,要考虑对已有索引的维护。这包括同步更新和异步更新两种方案。

同步更新要求索引的维护需要严格与数据的更新同步,如图 5 以更新数据项为例,在返回客户端更新成功的时候,索引的值就是最新的。同步更新适用于对索引与数据之间一致要求较高的场景,如一些金融场景的业务。

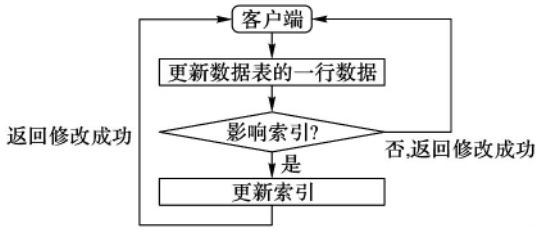

图 5　索引与数据的同步更新

相比同步更新,异步更新通过延后索引更新的时机来提高系统的更新性能,即更新操作不需要表和索引完全同步就可以成功返回客户端。仍以更新操作为例,如图 6 所示,只要保证在下一次读取到更新值之前的时间窗口内完成索引的更新,这样最终也能够保证索引和数据之间的一致。异步更新适用于索引与数据一致的要求不高的互联网应用。

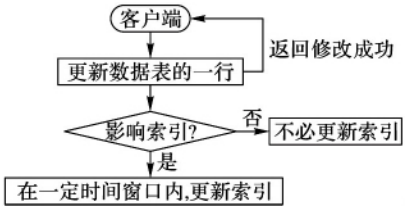

图 6　索引与数据的异步更新

3　OceanBase 概述

OceanBase[10] 是阿里集团研发的分布式关系数据库,其目的是以廉价服务器集群存储海量结构化数据并实现高可用、高可扩展性,即既支持跨行跨表的事务,又支持存储节点线性扩展,目前已经在淘宝、天猫以及支付宝等金融级业务中实践应用。本文基于 OceanBase0.4 的开源版本设计实现了分布式索引。

3.1 OceanBase 架构

OceanBase 0.4 的整体架构如图7所示，OceanBase 支持使用 Java 数据库连接(Java Data Base Connectivity，JDBC)和C客户端访问。RootServer 为 OceanBase 集群的主控节点，管理集群的所有服务器，并负责数据分布及副本的管理；UpdateServer 负责事务执行，并存储更新数据；ChunkServer 负责数据存储，通常采用多副本冗余技术来实现分区容忍性；MergeServer 则负责接收并解析用户 SQL 请求，最终将查询结果返回用户。在 OceanBase 集群部署时，通常将 RootServer 和 UpdateServer 部署到同一台服务器上，MergeServer 和 ChunkServer 也共享同一台服务器。

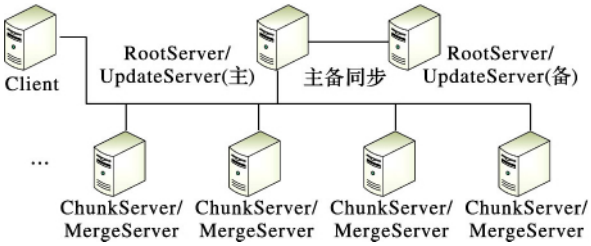

图7 OceanBase 架构

3.2 OceanBase 存储引擎及数据访问处理

OceanBase 数据库采用类似基于日志结构的合并树(Log-Structured Merge Tree，LSM Tree)[17]的内外存混合存储的模式，将数据的增量更新写入内存，而使用外存来存储(只读)基线数据。内存中的增量更新全由 UpdateServer 维护，因此 UpdateServer 的内存大小及处理性能要求较高，可以将 UpdateServer 视作提供了一个内存数据库。

如图8所示，系统对数据的增量修改单独保存在一台服务器内存中(相关日志会物化到磁盘)，内存中的数据增量称为 MemTable(内存表)，增量数据只存储在 UpdateServer 中。将每张表的基线数据按照行主键排序并划分为多个数据分片，称为 Tablet，并采用分布式 B+树的形式将 Tablet 存储在集群计算节点 ChunkServer 中，物化的文件称为 SSTable，SSTable 由多个数据块组成。根据内存使用状况以及定时机制，系统会不断将基线数据和增量更新融合，这个过程称之为合并。

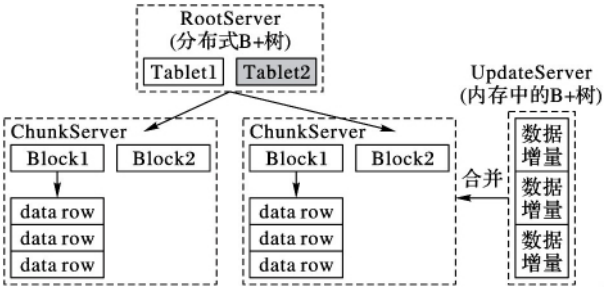

图8 OceanBase 基线数据和增量数据

数据以基线数据加增量修改作为查询的返回结果。如图9，MergeServer 解析查询语句，生成相应的执行计划，发送请求给对应的 ChunkServer，ChunkServer 收到执行计划，向 MemTable 请求数据增量更新，将最终结果返回。

在响应写事务时，由 MergeServer 解析 SQL 生成物理执行计划，并将执行计划发送给 UpdateServer 执行修改内存表 MemTable，如图10。其中，写事务可能需要读取基线数据用以判断更新数据行是否存在，这些数据也会一起随着执行计划发送给 UpdateServer。

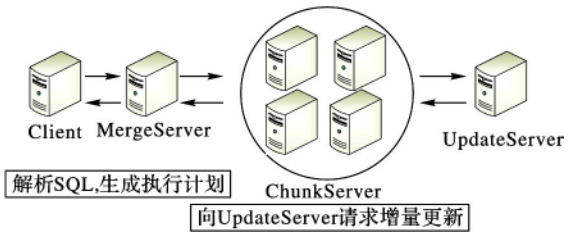

图9 OceanBase 访问数据流程

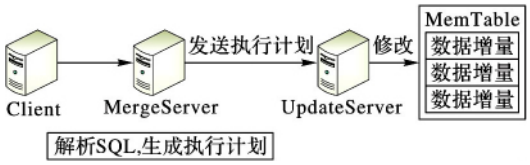

图10 OceanBase 更新数据流程

UpdateServer 执行物理计划并修改内存表 MemTable。MemTable 由索引和行操作链组成，索引以内存 B+树实现；行操作链表记录以列值 cell 为单位的不同版本的修改操作，包括未提交部分和已提交部分。如图11所示，修改 MemTable 时，由执行事务的线程锁住内存表的数据行，将对该行数据的操作追加到该行的未提交链表当中，然后往提交队列里加入一个提交任务，这个过程称作预提交，多线程并发执行。然后，再用一个提交线程从任务队列里扫描提交任务并写日志，将日志发送到备机后，将未提交操作链表中的 cell 操作追加到已提交链表末尾，再释放锁回复客户端操作成功。

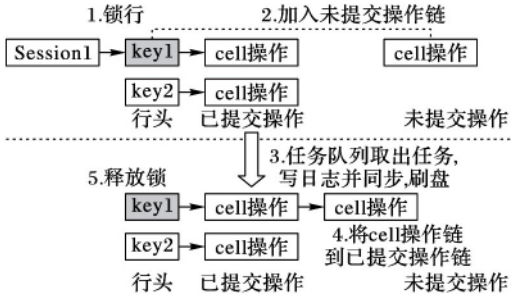

图11 OceanBase 修改内存表流程

4 OceanBase 索引的实现

4.1 索引的设计

在本文的实现方案中，将索引视为另一种数据，以表结构的形式创建并维护分布式索引，原因有以下两点。

1) OceanBase 是已经应用于涉及在线交易和金融业务的分布式存储系统，因此对索引的可用和稳定的要求较高。将索引以表的形式组织，能实现与数据相同的负载均衡和备份容错。

2) 采用表的方式组织和存储，可以让系统在对已有数据构建索引时，如果发生对数据的更新操作，能够将相应的索引更新利用读写分离的方式维护起来(保存在 MemTable 当中)，如图12。

本文的方案将索引列和原数据的主键作为联合主键存成索引表。实现分布式环境下的全局索引，采用全局索引的原因如下：

1) 索引分布信息的维护。选择以表的形式组织索引，

OceanBase 就能通过 RootServer 获取索引表 Tablet 在集群中的分布情况。

2) 较小的索引维护成本。因为 OceanBase 对写事务是由 UpdateServer 单点执行的,修改索引表 MemTable 造成的网络交互较小,维护全局索引的网络开销也较小。

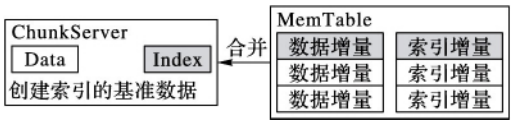

图 12　在内存表中维护索引的更新

在索引维护的设计上,本文在集群所有节点的索引构建完成之后,才使索引可用于服务。每个节点会根据需要创建索引的基准数据 SSTable; 同时将索引更新维护在 MemTable 当中。考虑到 OceanBase 主要用于在线交易和金融业务,其索引应当与数据同步更新,如图 12,数据和索引的增量更新同时写入内存表,这也符合 OceanBase 单点执行写事务的特点。

4.2　索引的实现与维护

在已有数据的状态下创建索引,前面已经提过 3 个要点: 集群系统需要根据既存数据创建索引; 在集群节点没有将既存数据的索引构建完成前,索引不可用于服务; 如果在创建索引期间,有对索引列的更新,要将这些更新也写入索引。根据 OceanBase 的数据读写分离特性,索引的创建流程如下:

1) ChunkServer 获取到需要构建的索引列的范围。

2) ChunkServer 读取本机上 SSTable 的数据; 同时去跟其他 ChunkServer 通信,获取构建全局索引需要的数据,写索引数据的 SSTable 文件。

3) 如果对数据的更新影响到了索引,需要将这些对索引的更新写入 UpdateServer 的内存表 MemTable。

4) 所有的 ChunkServer 的全局索引信息构建完成后,将索引表用于服务。

此流程实际上将全局索引分为两部分来维护: 1) 索引在 MemTable 的增量更新; 2) 索引在 ChunkServer 存储的基线数据。

在 ChunkServer 上通过实现 IndexBuilder 创建索引的基线数据。如图 13,IndexBuilder 主要实现以下流程:

1) IndexBuilder 为本机上的数据写一个局部索引 SSTable,按照索引列及主键联合排序。这是因为非主键访问数据性能不高,为了高效创建全局索引需要进行排序。

2) 排序完成后,IndexBuilder 发送本机上的数据采样信息给 RootServer,RootServer 根据采样信息决定全局索引的分布,返回给 IndexBuilder 需要构建的索引列的范围,记为 Range。

3) IndexBuilder 根据得到的 Range 与其他 ChunkServer 通信,从步骤 1) 中局部排序的 SSTable 中读取到 Range 范围内的数据。

4) 根据本机和从其他节点获取到的数据构建全局索引。

当拥有索引的数据被更新时,其索引信息也要同时被更新,才能保持数据和索引之间的一致。OceanBase 是由 UpdateServer 单点执行写事务,所以索引和数据的更新维护都是由 UpdateServer 来处理,可以实现索引和数据的同步更新。

为了实现索引和数据的同步更新,在 MergeServer 构建执行计划的时候,将对索引的更新写成一个包含索引信息的类(称为 Trigger)加入到执行计划发给 UpdateServer。UpdateServer 执行接收到的计划时,如果需要处理 Trigger,会在 MemTable 当中对索引也进行相应的更新修改,保证对索引和数据的更新在同一个事务当中,能够同时成功或者回滚。

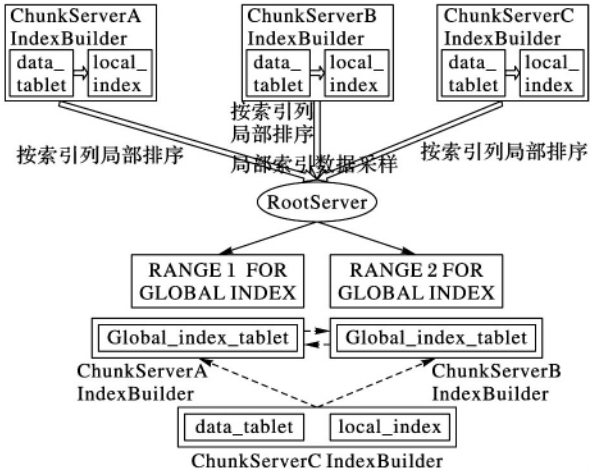

图 13　创建索引静态数据

如图 14 所示,UpdateServer 在执行物理计划处理 Trigger,修改 MemTable 时,会在同一个 session 阶段里将对数据和所有相应索引的更新操作加到 Memtable 提交链表,之后,执行 3.2 节的所述事务提交流程,在事务提交的阶段将两部分更新同时提交,保证二者的同步一致。无论对数据的修改需要维护多少索引,其更新都是同时提交的。

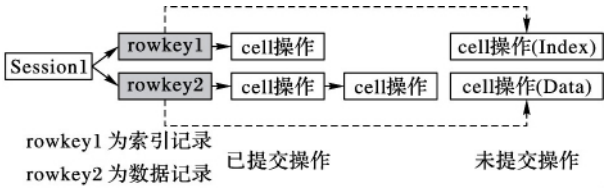

图 14　在 MemTable 中同时更新索引和数据

OceanBase 架构对 UpdateServer 的内存和处理性能要求较高,UpdateServer 可视作内存数据库,且仅维护每日的增量更新的数据,不用处理分发请求、SQL 解析等其他冗余工作,每日合并过后其内存可被释放,因此可以较好地进行单点处理写事务和索引更新。

4.3　索引的分布

决定索引在集群分布状况的要素有两个: 1) 索引的分片原则; 2) 索引表的冗余备份。OceanBase 会为存储的数据实现负载均衡和数据迁移工作,将索引存成表的方案可以很好地利用这一特性满足上述的第二要点。IndexBuilder 只创建一个备份的索引表数据,完成之后,RootServer 根据集群的负载状况,再选择其余的 ChunkServer 对索引表数据进行拷贝和迁移,使得索引表 SSTable 在 ChunkServer 上的分布满足负载均衡,也使得在访问数据的时候,各个节点不会处理过多的读取静态数据的请求。需要注意的是,这样的索引创建策略对其他的分布式存储系统也有很好的适用性,因为每个分布式存储系统都会实现每个存储节点的自动负载均衡工作。

本文索引的分片设计满足如下原则: 索引表表切分后的每个分片的数据量均衡,并且 SSTable 数接近主表的 SSTable

数目，过少则 SSTable 文件太大，影响静态数据每日合并，而产生过于琐碎的 SSTable 则会造成网络交互成本提高，影响数据访问性能。如 4.2 节描述，本文设计中 RootServer 根据 IndexBuilder 的数据采样信息决定数据分片。IndexBuilder 在对数据 SSTable 按照索引列及主键联合排序时采样，样本的信息为排序时的文件的部分数据的起始和结束的索引表主键，以及满足该主键范围的记录数目。如图 15，每隔 N 行记录下索引列以及原表的主键以及记录数 N 作为采样信息。

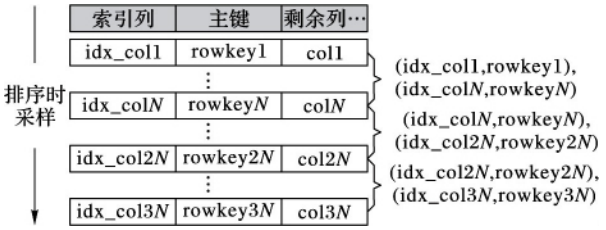

图 15 排序时记录索引表主键采样

N 值越小，采样信息越详细，索引数据切分后的分片越均衡，但传输样本的网络开销也随之上升，因此，通常根据数据表索引列的分布进行预估，再决定 N 的取值。

4.4 索引的正确性

因为原数据中的主键是唯一的，所以将索引列和原数据的主键作为联合主键对应的数据记录也有唯一性，在处理等值查询和范围查询时，都能根据索引列得到正确的原数据主键，从而按照 2.2 节中所述的全局索引查询流程得到正确的数据记录。

同时，如 4.1 节所述，本文的设计中，在创建索引的阶段同时能够在内存表中维护索引的更新，且索引和数据的更新在同一个事务中处理，保证一同成功或者回滚，这就保证了索引和数据之间的一致。

考虑到分布式存储系统的节点可能发生磁盘错误等故障，以及其他事故会导致索引表与数据表不一致，本文的设计为索引表与数据表实现了列校验和检查，即对数据表和索引表的对应列值进行循环冗余校验码（Cyclic Redundancy Check，CRC）校验计算，得到每列的列校验值（Column CheckSum）。列校验和在每日合并以及创建索引的阶段都会重新计算并比较检查，每个 SSTable 都会记录自己文件的列校验值，检查正确性时将数据和索引的所有 SSTable 对应列的列校验值相加并比较，通过 CRC 校验和确保索引与数据一致。

5 实验结果与分析

5.1 实验环境

本文使用 4 台虚拟机组成的集群作为测试环境，每台虚拟机的配置相同，包括 4 核 2.27 GHz 主频 CPU、36 GB 内存、99 GB磁盘，虚拟机上安装了 Red Hat Enterprise Linux Server release 6.2（Santiago）系统，相互之间通过千兆以太网连接。集群中的一台虚拟机被配置为 RootServer 和 UpdateServer，另外三台虚拟机被配置为 MergeServer 和 ChunkServer。集群中所有数据都被配置为 3 个副本。为了便于区分，在下文中用 OB 代表原始的 OceanBase 开源版本，用 OBi 代表增加了辅助索引功能版本。OBi-0 表示使用的是增加了辅助索引的版本，但未在数据表上创建任何辅助索引，OBi-1 则表示在数据表的一个非主键属性上创建了一个辅助索引，依此类推。

本实验中使用了雅虎公司开发的测试工具 YCSB（Yahoo Cloud Serving Benchmark）[18]，通过 JDBC 接口来访问 Oceanbase 集群。YCSB 所提供的核心负载集（Core Workloads）运行在测试表 Usertable 上，该表有一个主键属性（YCSB_PRIMARYKEY）和 10 个非主键属性（FIELD1 ~ FIELD10），所有属性均为 varchar 类型。YCSB 的核心负载集中包含有 6 个负载，分别对应于频繁更新（Update Heavy）、读为主（Read Mostly）、只读（Read Only）、读最近插入（Read Latest）、范围查询（Short Ranges）和读-修改-更新（Read-Modify-Write）6 个不同的场景。由于 OceanBase 不是针对范围查询的场景所设计，其范围查询的性能较差，本文未使用范围查询负载。此外，为了能够使用 YCSB 测试非主键属性查询的性能，本文对 YCSB 的源代码进行了修改，使之能够支持非主键属性的查询。

5.2 索引对 OceanBase 性能的影响

为了维护索引与数据表的一致，在更新操作时必然会有额外的开销。如图 16 本文测试了辅助索引对系统吞吐率的影响，其中 Load、Update Heavy 以及 Read-Modify-Write 这 3 个负载场景涉及到了对数据和索引的更新，从实验结果可以看出，在创建了索引的前提下，不同索引数目下的系统更新性能是相仿的，在索引数目 1 ~ 5 的 Load 和 Update Heavy 负载场景下系统吞吐率都是在 1 800 OPS（Operations Per Second）左右，而 Read-Modify-Write 负载场景下系统吞吐率均是在 1 300 OPS 左右。这是因为 UpdateServer 单点处理写事务，并且所有索引的维护都与数据更新放入同一事务执行，因此，选择全局索引方案，使索引的更新也通过单点执行是有成效的。实验结果还表明，在 OceanBase 上实现此索引方案，没有对原本的基于主键的数据读取访问性能带来影响，Read Modify、Read Only 以及 Read Least 的负载下系统吞吐率均维持在 2 000 OPS 左右。

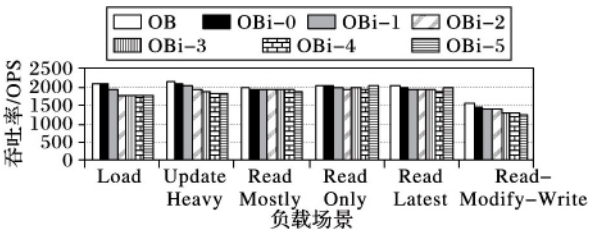

图 16 辅助索引对系统吞吐率的影响（10 线程/10 万条记录）

本实验还测试了不同数据规模下系统的吞吐率，如图 17 所示，随着数据规模从 10 万（10w）扩大到 1 000 万（1 000w），本文观察创建索引前后的系统的吞吐率变化，可以看出每个负载场景中吞吐率的变化值是比较稳定的，不同数据规模下 Load 和 Update Heavy 负载的吞吐率下降值都在 100 OPS 以内，而 Read-Modify-Write 的下降值都在 200 OPS 以内；而原本基于主键的数据访问没有受到影响。

表 1 记录了各个数据规模下，图 17 中以更新为主的负载场景的吞吐率下降情况，可以看到，Load 和 Update Heavy 负载场景的吞吐率下降稳定在 5% 以内，而 Read-Modify-Write 负载下的吞吐率下降则稳定在 12% 左右。

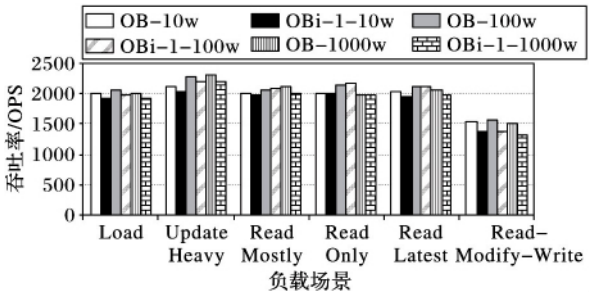

图 17　不同数据规模场景下的系统吞吐率(10 线程)

表 1　不同数据规模场景下更新负载吞吐率下降比例　%

负载场景	数据规模		
	10 万	100 万	1 000 万
Load	4.60	4.20	4.20
Update Heavy	4.09	3.40	5.00
Read-Modify-Write	11.10	12.37	12.30

未使用索引进行非主键查询时,因为读事务执行时间过长,超出了响应时间的限制,YCSB 无法完成各个负载场景的读性能测试。这是 OceanBase 在没有索引访问数据时需要对全部数据进行扫描再过滤数据的缘故。

本文测试了创建索引后不同选择率(即每个非主键查询能得到结果的数据量在测试数据中所占的比例)条件下的非主键查询性能,除数据规模的变化外还增加了负载场景,10w-storing 和 100w-storing 分别表示 10 万和 100 万的数据规模下,使用冗余了查询属性的索引表访问数据。此时数据访问可以直接从索引表中得到结果,如 2.1 节所述。

如图 18 所示,选择率的上升,意味着一次非主键访问返回的结果记录的数据量增大,因此,系统的吞吐率会显示下降趋势。使用了冗余查询列的索引表,由于减少了一次对数据表的查询,其非主键访问的性能高于不使用冗余查询列的索引。如在 100 万数据规模,5‰的选择率的负载下: 不使用冗余列的索引,吞吐率约为 200 OPS; 使用冗余列的索引,吞吐率约为 400 OPS,性能优化了 100%。而在 100 万数据,1%的选择率的负载场景中: 使用没有冗余查询属性的索引访问数据,由于执行时间过长,超出了 OceanBase 响应时间的限制,导致读事务失败; 而使用冗余查询属性的索引访问数据仍然可以执行完毕得到结果。

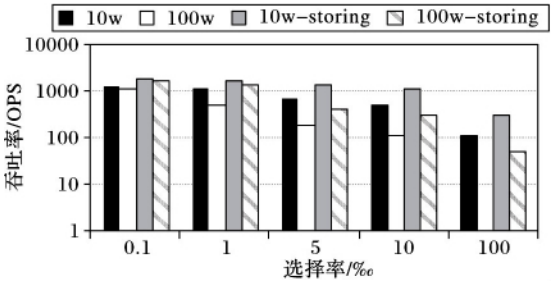

图 18　辅助索引在不同选择率条件下的非主键查询性能

6　结语

本文分析了为新的分布式存储系统设计辅助索引时要考虑的关键技术,基于系统特征、存储特点以及应用需求充分地考虑相应的索引组织形式、创建和维护方案,并在这些分析结论的基础上,为新型的分布式存储系统 OceanBase 设计实现了辅助索引机制。使用 YCSB 的测试实验结果表明,实现的索引在提高非主键查询性能的同时,对系统造成的性能影响是能稳定在一定范围的,而且在索引中冗余查询列可以进一步优化查询性能。

参考文献:

[1] CHANG F, DEAN J, GHEMAWAT S, et al. BigTable: a distributed storage system for structured data [J]. ACM transactions on computer systems, 2008, 26(2): Article No. 4.

[2] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: Amazon's highly available key-value store [C]// SOSP'07: Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles. New York: ACM, 2007: 205 - 220.

[3] CALDER B, WANG J, OGUS A, et al. Windows azure storage: a highly available cloud storage service with strong consistency [C]// Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. New York: ACM, 2011: 143 - 157.

[4] The Apache Software Foundation. Apache HBase [EB/OL]. [2015-06-09]. http://hbase.apache.org/.

[5] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system [C]// Proceedings of ACM SIGOPS Operating Systems Review. New York: ACM, 2010: 35 - 40.

[6] KARGER D R, LEHMAN E, LEIGHTON F, et al. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide Web [C]// Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing. New York: ACM, 1997: 654 - 663.

[7] Basho Technologies, Inc. Riak: a decentralized datastore [EB/OL]. [2015-06-08]. https://github.com/basho/riak.

[8] DATAR M, IMMORLICA N, INDYK P, et al. Locality-sensitive hashing, scheme based on p-stable distributions [C]// Proceedings of the 20th Annual Symposium on Computational Geometry. New York: ACM, 2004: 253 - 262.

[9] STOICA I, MORRIS R, KARGER D, et al. Chord: a scalable peer-to-peer lookup service for Internet applications [C]// Proceedings of the 2001 ACM SIGCOM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM, 2001: 149 - 160.

[10] Huawei-Hadoop. Hindex—secondary index for HBase [EB/OL]. [2015-06-20]. https://github.com/Huawei-Hadoop/hindex.

[11] GitHub, Inc. Indexed HBase [EB/OL]. [2015-06-10]. https://github.com/ykulbak/ihbase.

[12] WU S, JIANG D, OOI B C, et al. Efficient B-tree based indexing for cloud data processing [J]. Proceedings of the VLDB endowment, 2010, 3(1/2): 1207 - 1218.

[13] Amazon Web Services, Inc. Amazon Elastic Computing Cloud (EC2) [EB/OL]. [2015-06-15]. http://aws.amazon.com/ec2/.

[14] ZOU Y, LIU J, WANG S, et al. CCIndex: a complemental clustering index on distributed ordered tables for multi-dimensional range queries [C]// Proceedings of the 7th IFIP International Conference on Network and Parallel Computing. Berlin: Springer, 2010: 247 - 261.

[15] TAN W, TATA S, TANG Y, et al. Diff-Index: differentiated index in distributed log-structured data stores [C]// EDBT 2014: Proceedings of the 17th International Conference on Extending Database Technology. Berlin: Springer, 2014: 700 - 711.

(下转第 12 页)

损失。通过与图4对比可知，数据的完整性损失程度与查询性能是呈正比关系的，越高的查询性能意味着更大的数据查询完整性损失。

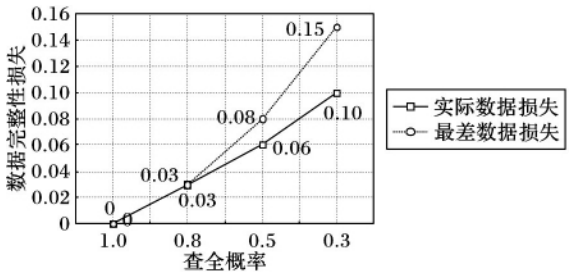

图6 查询误差

3.3 演示方法

Probery是基于Hadoop平台实现的大数据概率查询系统，用户需要以提交Hadoop作业的形式来运行系统。Probery总共为用户提供了两个操作接口，分别是数据装载和概率查询，当系统运行后，用户可以根据需求选择相应的操作。

在进行系统演示时，由于Probery运行环境的限制，对于上述两个Probery的基本功能，计划采用动画的形式对系统的这两个功能进行演示，并辅助以幻灯片的形式对系统的整个运行机制进行说明。

4 结语

本文阐述了Probery的系统架构、关键技术以及系统演示。实验结果证明，Probery在损失数据查询完整性的同时能够提高数据的查询性能，而且伴随着数据量的增加，Probery的数据查询性能稳定。

当然，Probery仍存在一些问题需要进一步的优化和完善，例如数据装载过程中数据的缓存机制、支持多种类的数据查询方式。

参考文献:

[1] The McKinsey Global Institute. Big data: the next frontier for innovation, competition, and productivity [R]. Chicago: McKinsey & Company, 2011: 5.

[2] EDLICH S. NoSQL definition [EB/OL]. [2015-09-15]. http://www.nosql-database.org/.

[3] Wikipedia. NoSQL. [EB/OL]. [2015-09-15]. https://en.wikipedia.org/wiki/NoSQL.

[4] 史英杰，孟小峰. 云数据管理系统中查询技术研究综述[J]. 计算机学报，2013，36(2)：209－225. (SHI Y J, MENG X F. A survey of query techniques in cloud data management systems [J]. Chinese journal of computers, 2013, 36(2): 209－225.)

[5] WHITE T. Hadoop: the definitive guide [M]. Sebastopol: O'Reilly Media, Inc., 2012: 9－12.

[6] Apache. Hadoop [EB/OL]. [2015-09-15]. http://hadoop.apache.org/.

[7] SHVAVHKO K, KUANG H, RADIA S, et al. The Hadoop distributed file system [C]// MSST 2010: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2010: 1－10.

[8] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107－113.

[9] GEORGE L. HBase: the definitive guide [M]. Sebastopol: O'Reilly Media, Inc., 2011: 27－29.

[10] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system [J]. ACM SIGOPS operating systems review, 2010, 44(2): 35－40.

[11] THUSOO A, SARMA J S, JAIN N, et al. Hive: a warehousing solution over a Map-Reduce framework [J]. Proceedings of the VLDB endowment, 2009, 2(2): 1626－1629.

[12] CHODOROW K. MongoDB: the definitive guide [M]. Sebastopol: O'Reilly Media, Inc., 2013: 3－5.

Background

This work is partially supported by the Major Program of the National Natural Science Foundation of China (61433008), the National Science Foundation for Young Scientists of China (61202088), the China Postdoctoral Science Foundation General Program (2013M5402 32), the Fundamental Research Funds for the Central Universities (N120817001), the Research Fund for the Doctoral Program of Higher Education of China (20120042110028).

WU Jinbo, born in 1992, M. S. candidate. His research interests include big data management, cloud computing.

SONG Jie, born in 1980, Ph. D., associate professor. His research interests include big data management, cloud computing, energy-efficient computing.

ZHANG Li, born in 1978, Ph. D., lecturer. Her research interests include big data management, service computing.

BAO Yubin, born in 1968, Ph. D., professor. His research interests include data warehouse, online analytical processing, cloud computing, data intensive computing.

(上接第7页)

[16] 阳振坤. OceanBase关系数据库架构[J]. 华东师范大学学报(自然科学版)，2014(5)：141－148. (YANG Z K. The architecture of OceanBase relational database system [J]. Journal of east China normal university (nature sciences), 2014(5): 141－148.)

[17] O'NEIL P E, CHENG E, GAWLICK D, et al. The Log-Structured Merge-Tree (LSM-Tree) [J]. Acta informatica, 1996, 33(4): 351－385.

[18] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB [C]// Proceedings of the 1st ACM Symposium on Cloud Computing. New York: ACM, 2010: 143－154.

Background

This work is partially supported by the Natural Science Foundation of Zhejiang Province (LY12F02044), the National Natural Science Foundation of China (U1401256).

WENG Haixing, born in 1993, M. S. candidate. His research interests include distributed database, big data management.

GONG Xueqin, born in 1974, Ph. D., professor. His research interests include database technique, social network analysis.

ZHU Yanchao, born in 1992, Ph. D. candidate. His research interests include distributed database.

HU Hualiang, born in 1974, Ph. D., associate professor. His research interests include data management.