云数据管理索引技术研究

马友忠 孟小峰

(中国人民大学信息学院 北京 100872)

摘 要 数据的爆炸式增长给传统的关系型数据库带来了巨大的挑战，使其在扩展性、容错性等方面遇到了瓶颈。而云计算技术依靠其高扩展性、高可用性、容错性等特点，成为大规模数据管理的有效方案。然而现有的云数据管理系统也存在不足之处，其只能支持基于主键的快速查询，因缺乏索引、视图等机制，所以不能提供高效的多维查询、join 等操作，这限制了云计算在很多方面的应用。学术界、工业界及各应用领域都对云数据管理中的索引技术进行了深入研究，提出了很多优秀的解决方案，本文主要对云数据管理的索引技术的相关工作进行了深入调研，并作了对比分析，指出了其各自的优点及不足；对我们在云计算环境下针对海量物联网数据的多维索引技术研究工作进行了简单介绍，最后指出了在云计算环境下针对大数据索引技术的若干挑战性问题。

关键词 云数据管理；索引；大数据

Indexing for Cloud Data Management

MA You-Zhong MENG Xiao-Feng

(School of Information, Renmin University of China, Beijing 100872

Abstract The explosive growth of the digital data brings great challenges to the relational database management systems, and they have troubles in scalability, fault tolerance etc. The cloud computing techniques have been widely used in many applications and become the standard effective approach to manage large scale data because of its high scalability, high availability and fault tolerance. While the existing cloud-based data management systems can’t support complex queries efficiently, such as multi-dimensional queries and join queries because of lacking of index or view techniques, these shortfalls limit the application of cloud computing in many respects. Many research works have been done in literatures, industrial and applications to explore the index techniques for cloud data management. In this paper we made in-depth research about the index techniques for cloud data management and pointed out their strengths and weaknesses. We also introduced our preliminary work on the index for massive IOT data in cloud environment. Finally we pointed out some challenges about the index techniques for big data in cloud environment.

Key words cloud data management; index; big data

1 引言

随着信息技术的飞速发展，在电子商务、物联网、社交网络、计算机仿真、科学计算、移动互联网等众多应用领域，数据量正在以指数级的速度增加，人类已经进入了大数据时代。

据统计，全世界的信息量每两年以超过翻倍的速度增长，2011 年将产生和复制 1.8ZB 的海量数据，其增长速度已经超过摩尔定律。传统的关系型数据库虽然能够提供十分成熟的数据存储、索引以及查询处理方案，然而面对不断增长的海量数据，关系型数据库在扩展性方面遇到了严重的瓶颈，无法实现高效灵活的扩展。虽然一些专业的公司能够提供一些针对关系型数据库的扩展方案，但是其部署、管理的代价非常大。

自 2004 年以来，Google 公司先后提出了 Google File System[1]、BigTable[2]和 MapReduce[3]等技术，随着这三大关键性技术的提出，云计算作为一种新的海量数据存储、管理、分析模式应运而生，并得到业界众多大公司的广泛应用和深入研究，云计算已经成了海量数据处理的一个标准解决方案。同时也产生了很多优秀的分布式数据存储和管理系统，如雅虎的 PUNUS[4]、Amazon 的 Dynamo[5]、开源的 HBase 等。基于 Key-Value 存储的云数据管理技术具有高可扩展性、高可用性和容错性等特点，能够实现对海量数据的高效存储和处理。

然而，现有的基于 Key-Value 存储的云数据管理系统在数据访问方面提供的功能比较简单。云数据管理系统大都是按照 rowkey 的顺序对数据进行组织，并在 rowkey 上建立类 B+tree 的索引结构，所以在 rowkey 上能够提供高效的点查询或范围查询。然而针对非 rowkey 的查询只能通过全表扫描的方式来实现。虽然我们可以利用 MapRedeuce 技术来实现数据访问的并行化，在一定程度上提高查询速度，但是当数据量非常大的时候，对于时间延迟要求比较高的应用来说，全表扫描所需的时间仍然比较长，无法满足实际应用的需求。

在实际应用中，除了对 rowkey 的查询之外，还有很多针对非 rowkey 的多维查询需求。如在基于位置的服务中，我们经常需要针对某个对象的经度、纬度、时间等属性进行多维查询；在图片共享服务中，我们可以对图片的拍摄时间、拍摄地点、图片主题等属性进行查询；在电子商务网站中，商品的数量往往达到数十亿、甚至上百亿，并且每件商品都有几十个甚至上百个属性，如名称、类别、价格、上架时间等。用户往往需要从多个不同的角度对商品进行查询，从而对所要购买的商品有更加全面深入的了解。

然而由于目前云数据管理系统在数据查询方面的局限性，限制了其在众多领域的广泛应用。索引是实现多维查询的一个有效方案，因此目前已有很多学者、公司针对云数据管理中的索引技术开展了大量研究工作，并提出了一系列有价值的解决方案。比如新加坡国立大学的 epiC 项目组创新性地提出了双层索引框架，并在此基础上给出了一系列解决方案；华为公司基于 HBase 的 coprocessor 技术设计了新的二级索引方案，大大提高了查询的效率。本文主要对云数据管理索引技术的相关工作进行了深入调研，分析了各自的优点及不足；并对我们在物联网索引方面的研究工作进行了简单介绍，最后指出了在云计算环境下针对大数据索引技术的若干挑战性问题。

本文主要内容如下：第二节主要对学术界关于云数据管理中的索引技术进行了归纳总结，并根据其采用的索引方案不同进行了分类；第三节主要对工业界及不同类型应用中的索引技术进行了分析，包括 NoSQL 数据库中的索引技术、海量数据处理中的索引技术、不同应用领域中的索引技术等，其中对轨迹数据、空间数据和图数据中的索引技术进行了详细分析；第四节主要是对我们在云计算环境下针对海量物联网数据的多维索引技术的研究工作进行了介绍；第五节指出了云计算环境下针对大数据的索引技术中存在的若干挑战性问题；最后对本文进行了总结。

2 云数据管理索引技术-学术界成果

2.1 概述

为了丰富云数据管理系统的查询功能，很多学者开展了云数据管理系统中索引技术的研究工作，目前已经提出了很多有价值的实现方案。我们对各种索引方案的索引结构、实现方式、优缺点进行了深入分析，并按照实现方式的不同对其进行了分类，主要有以下五种不同的方案：分别为双层索引方式、二级索引方式、全局分布式索引、基于线性化技术的索引和基于位图的索引。表 1 给出了各种索引方案的优缺点、主要索引结构及代表性技术。

表 1 各种索引方案对比

索引方案	优点	缺点	主要索引结构	代表技术
双层索引	可以通过全局索引来定位数据所在节点，避免额外查询开销	索引维护代价高； 实现较复杂； 维度较高时，查询性能急剧下降	B-tree	B-tree + BATON[6]
			R-tree	RT-CAN[7] A-Tree[8] EMINC[9]
			Quad tree	QT-Chord[10]
			Bit map + R-tree	TLB-index[11]
二级索引	实现简单； 维护代价低	多维查询效率较低	索引表	ITHbase IHbase Asynchronous View[12]
		空间冗余大	索引表	CCIndex[13]
全局分布式索引	实现简单； 自动负载均衡	客户端负担较重； 无法满足数据频繁插入	B-tree	Distributed B-tree[14]
基于线性化技术的索引	较高的写入吞吐量；维护代价较低	一致性维护复杂	KD-tree	MD-HBase[15]
位图索引	高并发； 避免额外检索代价	属性值域划分规则复杂； 范围查询时会出现很多 false positives	Bitmap	RBI[16]

索引的主要目的是尽可能避免不必要的数据扫描和比较，快速定位到所要查找的数据。在云数据管理系统中，数据一般都是以 key-value 的形式进行存储的，并且按照 rowkey 对数据进行组织、分块，在 rowkey 上建有相应的索引结构，所以系统能够提供针对 rowkey 的高效查询。而对于非 rowkey 上的查询则往往需要通过全表扫描的方式来进行逐一比较，效率较低。因此索引的主要目的就是提高针对非 rowkey 的查询速度。

第一种方案和第二种方案索引的数据对象可以是分布式文件系统（GFS、HDFS）中的数据，也可以是 key-value store（Bigtable、HBase）中的数据，索引只是负责对已经进入到系统中的数据建立索引，不影响或控制数据的分布及存储。而第三种方案和第四种方案略有不同，这两种方案在建立索引的同时，也会影响数据的分布。其中，第三种方案是以分布式 B-Tree 的方式对数据进行组织，新来的数据，根据 B-Tree 的插入规则，找到相应的插入节点，并根据具体情况，对节点进行适当分裂。B-Tree 中的节点（包括内部节点和叶子节点）分布到各个不同机器上。第四种方案主要是采用空间目标排序技术把多维数据映射到一维空间上，并以一维空间中的数值作为记录的 rowkey。同时利用 KD 树或四叉树对多维数据空间进行划分，根据最长公共前缀的方式计算得到每个子空间的名称，并以此名称作为索引项对数据各个子空间的数据进行索引。

2.2 双层索引

目前已有的云数据管理索引方案中关于双层索引的研究比较多，研究人员主要来自新加坡国立大学、中国人民大学、东北大学、希腊 Thessaly 大学。双层索引方案主要是为了满足数据的海量性以及系统的可扩展性而提出来的。伍赛等人[17]于 2009 年提出了一个云数据管理系统中的索引框架，后续的关于双层索引方案的研究大都是基于该框架进行的。该框架的实现方案如图 1 所示：

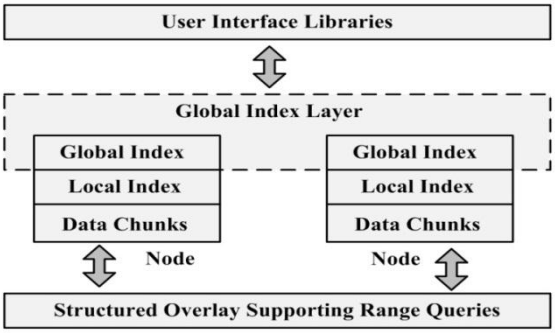

图 1 双层索引基本框架

由上图可知，索引部分包括局部索引和全局索引两部分。在云数据管理系统中，有大量的廉价计算机组成的计算机集群可以为用户提供计算资源和存储资源。用户数据按照一定的规则被划分成数据块，这些数据块按照分布式文件系统的协议被分配到不同的计算机节点存储。在双层索引方案中，对每个计算机节点的数据建立一个本地的局部索引，该局部索引只负责本地节点上的数据。除局部索引外，每个计算节点还需要共享一部分存储空间用来存储全局索引。全局索引是由部分局部索引组成的，由于存储空间的限制和查询效率的要求，不可能把所有的局部索引节点全都发布到全局索引中，所以针对局部索引需要按照一定的规则选择其中的一部分索引节点来进行发布。对于被选中的索引节点，在全局索引中可以有不同的方案对其进行组织。

在双层索引方案中，根据局部索引的索引结构以及全局索引的组织方式不同，又可以分为多种不同的种类，表 2 描述了各种方案的局部索引、全局索引以及节点选择策略。

表 2 双层索引方案对比

方案名称	局部索引	全局索引	索引节点选择策略（代价模型）
Efficient B-tree[6]	B^+-tree	BATON	Query cost + Index maintenance cost
RT-CAN[7]	R-tree	CAN	Query cost + Index maintenance cost
QT-Chord[10]	IMX-CIF quad-tree	Chord	Naming function of block code
EMINC[9]	KD-tree	R-tree	Index node volume
A-Tree[8]	R-tree + Bloom filter	Array Based	Index node volume
TLB-index[11]	B^+-tree for Bitmap	BATON for Bitmap	Not explain in detail

Efficient B-tree[6]是一个基于的双层索引方案，主要支持单个属性上的点查询和范围查询。在每个存储节点上，针对本地数据建立相应的 B^+-tree 索引，然后按照特定的索引节点选择策略从每个局部索引中选择一部分节点发布到全局索引中。为了提高查询的速度，保证系统的可扩展性，全局索引采用了 P2P 网络 BATON[18]对全局索引节点进行组织。在进行索引节点选择的时候采用了基于代价模型的自适应调整策略，作者提出了一个代价模型，每个节点的代价包括查询代价和索引维护代价，在进行节点选择的时候保证总的代价最小。具体调整过程是：最初选择每个局部 B^+-tree 索引的根节点发布到全局索引中，然后定期地计算全局索引节点的子节点的代价之和，如果子节点的代价之和小于原有父节点的代价，那么就把原有的父节点替换成新的子节点。该方案能够对单个属性的查询提供高效的支持，但是无法实现多维查询。

RT-CAN[7]的基本实现思路与 Efficient B-tree[6]是一样的，不过 RT-CAN[7]是针对多维查询而设计的。为了支持多维查询，RT-CAN[7]在局部索引中采用了 R-tree 结构，在全局索引中采用了能够支持多维查询的覆盖网络 CAN 网络，节点选择策略和 Efficient B-tree[6]基本一致。 R-tree 是一种类似于 B^+-tree 的索引结构，都是一种平衡树，并且节点的散度比较大，树的层次比较少，所以在查询的时候效率比较高。但是为了保持树的平衡，在数据插入的过程中，需要不断地对索引节点进行分裂、调整，所以索引的维护代价比较高，尤其是对数据插入比较频繁的应用来说，索引的维护代价过高，会对系统的吞吐量带来一定的影响。因此，文献[10]提出了另外一种双层索引结构 QT-Chord。在 QT-Chord 中，局部索引采用的是改进的四叉树 IMX-CIF Quad-tree, 全局索引采用的 Chord 覆盖网络[19]。在索引节点的选择上，QT-Chord 并没有利用基于代价的模型，而是基于四叉树的编码方法对每一个四叉树节点进行编码。由于父节点的编码是子节点编码的前缀，所以作者充分利用这一特点，设计了一个命名函数，把子节点递归地映射到上层父节点，然后把最后映射到的节点发布到全局索引中。

前面三种方案在全局索引中均采用了基于 P2P 结构的覆盖网络，这种方式能够很方便地实现系统的可扩展性，使系统能够同时支持大规模的查询。但是这种方式也存在一些不足之处，首先 P2P 网络本身需要一定的维护代价，在查询的时候，往往也需要一定的网络传输代价，这在一定程度上会增加系统的延迟；另外，由于现有的云数据管理系统一般都是 master-slave 结构的，要在这些节点上重新构建一个 P2P 网络，会对原有的存储系统带来一定的负面影响。基于这些原因，EMINC[9]和 A-Tree[8]在全局索引中采用了集中式的索引方式。EMINC[9]中针对每一个局部节点建立一个 KD-tree[20]，然后选择 KD-tree 的部分节点作为全局索引，作者把每一个局部索引节点看成是一个多个维度的立方体，然后在全局索引中用 R-tree 对这些立方体进行索引。由于 R-tree 树的各个节点之间会有一定的重叠，尤其是当维度比较高，或者数据量比较大的时候，重叠部分会比较多，这样在查询的时候，尤其点查询，就会造成大量的 false positive 查询，从而影响查询的效率。为了解决这一问题，A-Tree[8]提出了另外一种方案–带 Bloom filter[21]的 R-tree。基本思路是这样的：针对每一个存储节点构建一棵 R-tree，同时创建一个 Bloom filer,这样，在进行点查询的时候，首先通过 Bloom filter 进行验证，如果查询点不在其中，则不再进行 R-tree 查询，否则继续进行 R-tree 查询。这样就减少了 false positive 的机率，从而提高查询效率。

TLB-index[11]采取的索引方案与 Efficient B-tree[6]采用的方案基本一致，局部索引用的是 B+-tree，全局索引用 BATON 网络，不同之处在于 TLB-index[11]对每一个不同的值构造一个 bitmap, 然后利用 B+-tree 对所有的 bitmap 进行索引，这样可以减少存储开销。

上述各种索引技术方案具有较好的扩展性，但是总体来说实现过程比较复杂，索引更新维护的代价比较高，对于数据更新比较频繁的引用来说，会影响系统的性能。在全局索引中大都采用了基于 P2P 协议的覆盖网络来实现并行查询，但是 P2P 网络本身的维护比较复杂，

查询时的网络开销也比较大，这会影响到系统的查询性能。

2.3 二级索引

二级索引（secondary index）方案类似于关系数据库中的二级索引，该方案主要应用于key-value 存储的云数据库管理系统中，如 BigTbale、HBase 等，目前大多数的二级索引方案都是基于 HBase 实现的。在 HBase 中，数据是以表的形式来存放的，为了实现对非 row-key 的查询，需要为每一个索引列构建一个索引表，以索引列的值（其实是索引列与 row-key 的组合）作为新的 row-key，实现索引列与原有 row-key 的映射。查询的时候，首先通过在索引表中进行查询，找到相应的原有 row-key 的列表，然后再根据这些 row-key 信息到原来的数据表中去查找所需要的原始数据信息。

目前基于二级索引的实现方案主要有 ITHBase、IHBase、CCIndex[13]和 Asynchronous views[12]。其中 ITHbase 和 IHBase 是两个开源的实现方案，ITHbase 主要关注于数据一致性，事务性是其重要特性。IHBase 与 ITHBase 比较相似，从 HBase 源码级别进行了扩展，重新定义和实现了一些 Server，Client 端处理逻辑，所以，它是具备强侵入性的。不过 IHBase 在修改完 HBase 0.20.5 版兼容 bug 以后没有再更新。

除了 ITHBase 和 IHBase 之外，中科院提出了另外一种二级索引方案：互补聚簇式索引（Complemental Clustering Index），简称 CCIndex[13]。前面的二级索引方案中，索引表中仅仅存放索引列与原来的 row-key 信息，在查询的时候，通过查询索引表得到 row-key 以后，还需要根据 row-key 到原来的表中去查找，然而由于得到的 row-key 大都是随机的，所以需要进行大量的随机读操作才能最终得到所需要的数据，效率相对比较低。为了减少随机查询带来的开销，CCIndex 提出了一个新的方案，把数据的详细信息也存放在索引表中，这样在查询的时候就可以直接在索引表中通过顺序扫描找到相应的数据，即把随机读变成了顺序读，可以大大减少查询时间。然而，把详细信息存储在索引表中会造成存储空间的增加，为了尽可能给地减少存储空间的开销，作者把 HDFS 文件块备份数设为 1，这样就可以保证存储空间不会增加太多。但是，备份数设为 1 之后，数据的容错性又成了新的问题，为了解决这一问题，作者又提出了新的容错方案，通过创建 clustering check table，再配合原有的索引表，能够实现快速恢复。同时，CCIndex 还提出了一种查询优化机制，用以支持多维查询。该优化机制主要是利用 HBase 中的一些元数据信息（region-to-server information）来估算每个子查询结果的大小，根据查询结果来生成合适的查询计划，从而降低查询时间。

CCIndex 方案实现起来相对比较简单，但是也存在一些不足之处，如存储开销比较大，尤其是当索引列比较多的时候，空间开销会更大；索引更新代价比较高，会影响系统的吞吐量；索引创建以后，不能够动态增加或修改。

Asynchronous View[12]提出了一种异步视图的方式来实现非 row-key 的查询，视图的方式类似于二级索引，因为视图物化以后就相当于索引。作者提出了两种视图方案：远端视图表（Remote View Tables: RVTs）和局部视图表（Local View Tables: LVTs）。RVTs 是指针对某个查询列建立一个视图，视图与原始表一样，可能会被分散存储到各个不同的节点上，这样视图中索引的数据与视图所在节点的数据可能不一致，当节点中的数据发生改变的时候，就需要对其他节点上的视图进行更新维护，代价较高。这种方式对于查询来说效率比较高，同时维护代价也比较大。 LVTs 是针对每个节点的数据分别建立一个局部的视图，这样节点数据发生改变的时候就可以直接在本地对局部视图进行更新，维护代价较低。但是在查询的时候，就需要对所有节点的视图进行查询，然后再进行汇总，所以查询效率比较低。一般来说，对于聚集查询，可以用 LVTs, 其他查询可以用 RVTs，针对一些特殊的查询，还可以采用两者相结合的方式。

总的来说，二级索引方案实现起来相对比较简单，但是会带来一定的存储开销，并且对

多维查询的支持不够理想。

2.4 全局分布式索引

为了支持大规模数据存储，并保证系统具有较高的吞吐量，Distributed B-tree[14]提出了一种容错的、高可扩展的分布式 B-tree 结构。该方案除了具有传统 B-tree 的一般特点之外，还具有一些新的特性：自动负载均衡、操作的原子性和存储节点的动态增加或删除。其基本思想如图 2 所示：

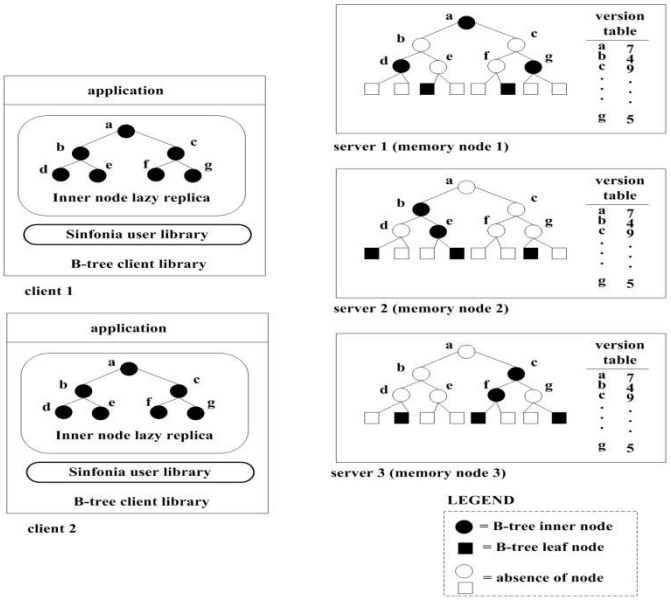

图 2 分布式 B-tree

所有数据以 B-tree 结构进行组织，B-tree 的节点（包括内部节点和叶子节点）分散存储在不同的节点上。为了实现数据的一致性，作者引入了 version table，用以记录节点的最新版本。为了提高查询的效率，B-tree 的索引内部节点都会缓存在客户端，并采用 lazy replica 的方式进行更新。这种方法主要有两个缺点：一是该方法对于简单的点查询具有较高的效率，但是对于复杂的范围查询和多维查询效率较低；第二个缺点是服务器端的维护代价较高、客户端需要消耗大量的内存空间去缓存 B-tree 的内部节点。

2.5 基于线性化技术的索引

上述几种索引方案均需要维护一种特定的索引结构， 如 R-tree 或 B-tree，当数据更新十分频繁的时候，索引更新维护的代价就非常高，所以为了降低索引更新维护的代价，同时又能保证系统的性能，MD-HBase[15]提出了一种基于线性化技术的索引方案。MD-HBase[15]的基本思想是：按照一定的规则将整个空间范围划分为大小相等的格子，并给每一格网分配一个编号，用这些编号为空间目标生成一组具有代表意义的数字。其实质是通过某种特定的方式将 k 维空间的实体映射到一维空间，从而可以利用现有数据库管理系统中比较成熟的一维索引技术来组织数据。用一维的数值对多维的空间目标进行排序的方法有很多，常见的方法有 Z-order 排序[22]、Hilbert 曲线等。这些技术的思路基本相同，都是利用一个线性序列来填充空间，构造一种空间填充曲线，把多维空间的数据转换成一维数据。这种转换关系必须能够较好地保持原多维空间目标之间的临近关系，从而提供较好的多维查询性能。

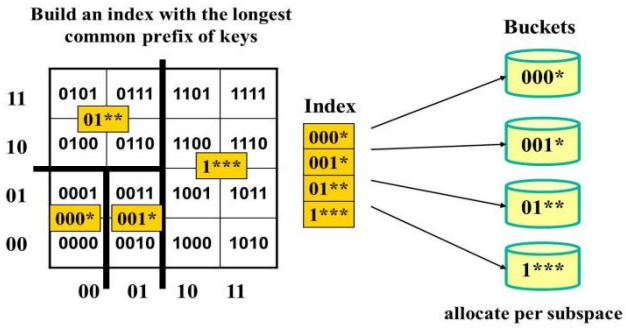

图 3 基于最长公共前缀的索引

MD-HBase[15]以 HBase 作为数据存储方案，用 Z-ordering 技术对数据进行排序，并以 Z-value 作为每条记录的 row-key 值。但是单纯的 Z-ordering 排序方法在搜索的过程中会带来一些额外的不必要的搜索空间（false positive search），所以，作者在此基础上利用 KD 树或四叉树对多维数据空间进行划分，根据最长公共前缀的方式计算得到每个子空间的名称，并以此名称作为索引项对数据各个子空间的数据进行索引，从而减少不必要的检索，大大提高了查询效率。基本思路如图 3 所示。

但是，该方法在进行空间划分的过程中，会带来数据的一致性问题，虽然目前有相应的解决方案，但是实现起来仍比较复杂，也会带来一定的额外负担。

2.6 分片位图索引

分片位图索引[23]是在结合集中式索引和分布索引各自优点的基础上提出的一种新的混合式索引方案。表 3 给出了集中式索引、分布式索引、扩展分布式索引以及混合式索引各自的优缺点。集中式索引的索引项一般由索引字段和原数据记录的主键组合而成，索引项按照索引字段值排序，同时索引项也采用 key-value store 来管理。索引集中式方案往往具有高可靠性、 可扩展性和 易管理性。但是由于查询以集中式方处理，因此无法通过并行化来充分利用分布式资源的优势。分布式方案是在各个节点建立局部索引，节点之间相互独立，因此局部索引的结构比较灵活，可以实现高度并发查询；但是由于每次查询都需要查询所有的节点，所以往往会带来额外的、不必要的查询开销，特别是对选择率很低的等值查询，会访问很多无用节点。扩展的分布式索引方案是在局部索引的基础上增加了一层全局索引，当一个查询来的时候，先通过全局索引来确定对应的局部索引，然后再到各个局部节点去查询，从而避免了不必要节点的查询。但是全局索引往往采用一些 P2P 网络如 BATON、CAN[24]网络来实现，结构比较复杂，网络开销往往比较大。分片位图索引[23]中作者也采用了全局索引和局部索引相结合的方式来实现，但是全局索引采用了一种比较新颖和有效的方式。相比以往扩展分布式方案中使用复杂 P2P 网络所实现的全局索引相比，分片位图索引采用基于属性值的全局排序机制，通过一个全局位图来表示局部数据在全局值域中得分布情况，从而为各数据节点提供一定的全局信息，并且该全局位图直接存储在各个节点上，因此不需要特殊的全局索引结构来实现，因此可以避免不同数据节点之间的网络通信开销。除全局索引之外，[23]还引入了位图索引的分片机制，即各局部节点也都有各自的局部索引，从而使得各数据节点的查询任务相互独立，便于并发执行，从而提高查询效率。但是该方案存在的一个问题在于，进行全局排序的时候，各属性值域的划分规则比较难以确定，因为不同的划分规则，对查询的性能会产生比较大的影响。除此之外，在实际应用当中，业务类型经常会发生改变，因此需要增加新的索引列，而要增加新的索引列，上述索引方案就需要对全部数据进行重新处理、索引，代价往往会非常大。

表 3 集中式索引 vs. 分布式索引

索引类型	实现方式	优点	缺点	代表技术	
集中式索引	索引项=索引字段+主键； 索引项按照索引字段值排序； 索引项也用 key-value store（e.g.,HBase）来管理。	高可靠性； 可扩展性； 易管理性；	查询以集中式方式处理； 无法通过并行化来充分利用分布式资源的优势； 响应时间较长；	二级索引	ITHbase CCIndex[13]
				基于线性化技术的索引	MD-HBase[15]
分布式索引	各数据节点各自建立局部索引； 没有索引值的全局排序信息，各节点相互独立；	可以实现高度并发查询； 局部索引结构灵活；	对选择率很低的等值查询，会访问很多无用节点； 降低系统的吞吐量；	二级索引	IHbase
				全局分布式索引	Distributed B-Tree[14]
扩展分布式索引	各数据节点各自建立、维护局部索引； 全局索引用于管理各局部索引；	可以在充分保证并行化的同时，减少不必要的检索代价	全局索引结构比较复杂； 全局索引的网络通信开销比较大；	双层索引	B-tree + BATON[6] RT-CAN[7] QT-Chord[10] EMINC[9]
混合索引	全局索引：索引字段上的全局排序； 局部索引：分片位图索引	既能充分发挥高度并行的优势，又能避免不必要的检索代价。	各属性值域的划分规则比较难以确定；	分片位图索引	RBI[23]

3 云数据管理索引技术-工业界及应用领域

文章第二部分主要介绍了目前学术界关于云数据管理中索引技术的研究成果，并对相关成果进行了分类、比较和分析。除了学术界的研究之外，工业界、云数据管理系统开发团队以及各个具体的应用领域，也针对云数据的索引问题进行了相应的研究。按照在数据管理中所处的层次不同，可以分为数据存储层、数据处理层和具体应用，如图 4 所示。在存储层主要包括分布式文件系统和 NoSQL 数据库中的索引技术，应用层主要包括在轨迹数据、空间数据、文本数据、图数据以及浮动车数据中的索引技术。下面将就这些问题详细介绍。

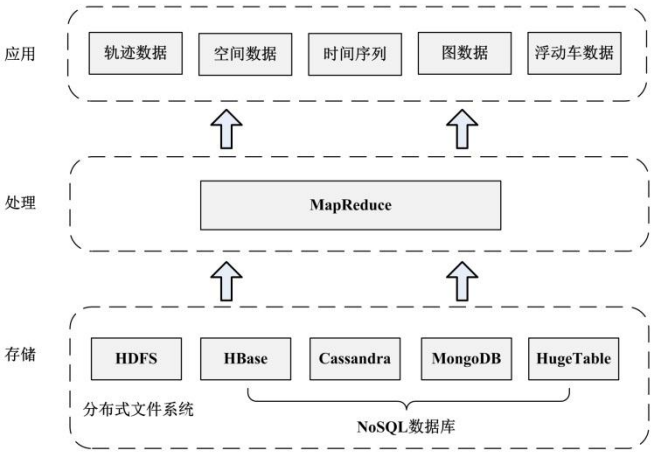

图 4 索引在不同层次中的应用

3.1 云数据管理系统中的索引技术

随着云计算技术的不断升温，云数据库也得到迅速发展，目前已经出现了很多云数据库管理系统，如 HBase、Cassandra、MongoDB 以及 Hadoop++[25]、HugeTable[26]等。这些云数据库系统都具有较好的可扩展性，为海量数据的存储和管理提供了一个行之有效的方案。但是现有的云数据库管理系统只能提供比较简单的基于 row-key 的查询，缺乏索引支持，无法提供复杂的多维查询、join 等操作，从而限制了其在某些方面的应用。针对现有系统存在

的上述问题，一些企业、科研人员以及各开源社区团队都进行了一些研究，设计了不同的索引方案，表 4 列出了几种常见的云数据库管理系统的索引方案，下面将详细介绍。

表 4 云数据库系统汇总

名称	索引方案	体系结构	备注
HBase	Secondary Index	Master/Slave	分布式的、面向列的开源数据库
Cassandra	Secondary Index	Peer to Peer	非关系型数据库
MongoDB	Support many index	Master/Slave	分布式文档数据库
HugeTable	Dense index, Sparse index and Block index	Master/Slave	海量结构化数据存储系统

3.1.1 HBase

HBase 初始版本对索引并不支持，后来逐步增加了索引模版，主要采用的是二级索引方案。二级索引方案的主要缺点是不能直接支持多维查询。在 HBase 的基础上曾经有两个开源的解决方案 ITHBase、IHBase，都可以提供索引支持，但是由于数据安全性、一致性等问题，后来两种解决方案都停止了版本更新，也没有得到广泛的应用。Coprocessor 技术出现以后，在 HBase 中就可以利用 coprocessor 所提供的一些特性，来安全地实现二级索引的功能，所以，现在 HBase 的版本中已经可以直接支持二级索引的创建。但是之前在 HBase 中实现的二级索引方案效率比价低，主要的原因在于存在大量的随机读，具体流程如下：当一个针对非 row-key 的范围查询来的时候，首先需要去查询对应的二级索引表，找到相应的 row-key 列表，返回到客户端，然后客户端再根据查到的 row-key 再到原来的数据表中去查找原始数据。由于查询到的 row-key 往往是随机、不连续的，所以客户端就需要向服务端发送多个查询请求，这样就会带来比较大的开销。

为了解决这一问题，一些研究机构基于 coprocessor 机制对 HBase 的二级索引进行了创新性的修改，如华为公司，其改进的重点在于减少由随机查询带来的大量网络开销。下面以 scan 操作来描述其具体的实现机制。

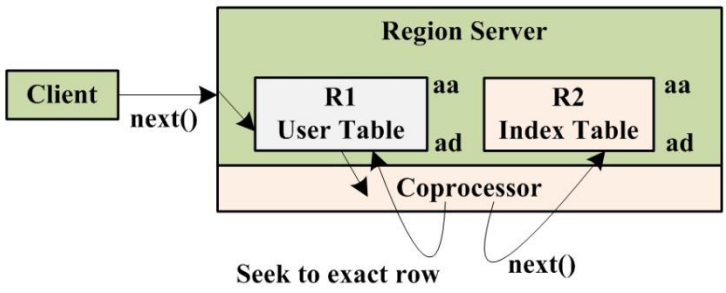

图 5 scan 操作流程

如图 5 所示，R1 表示原用户表的一个 region，R2 是索引表的一个 region，并且 R2 中的索引的数据跟 R1 中的数据是相互对应的，即 R2 中只索引 R1 中的数据，这是跟原有二级索引的一个主要区别。当客户端发起一个 scan 操作的时候，coprocessor 会去查询各个 region 对应的索引表的 region，通过索引表可以查到原有数据的 row-key 列表，然后再根据得到的 row-key 列表去 R1 中查询原始数据的详细信息，最后再把结果返回到客户端。

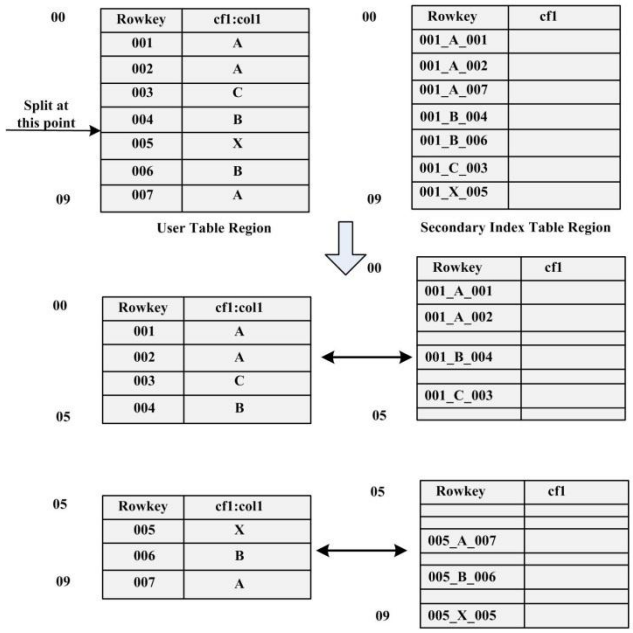

图 6 region 分裂机制

由图 6 可以看出，在最终结果返回之前，所有的操作都是在服务端完成的，所以就减少了大量的网络开销，从而提高了查询效率。

为了达到上述目的，在建立索引表的时候，必须采取特殊的机制。由于要保证索引表的 region 要与原数据表的 region 一一对应，所以索引表不能够自动分裂，其分裂必须由原数据表来控制。如图 6 所示：当原数据表的一个 region 分裂成两个子 region 的时候，其对应的索引表的 region 也分成两个新的子 region，并且和数据表的两个子 region 一一对应。

上述实现方案具有很多优点，如可以在多个列上建立索引，并且可以动态地增加或减少索引，同时不需要对 HBase 的核心代码做太多的修改，便于 HBase 的升级和维护。该索引实现方案应该是目前为止扩展性和实用性都比较好的一种方案。

3.1.2 Cassandra

Apache Cassandra 是一套开源分布式 Key-Value 存储系统。它最初由 Facebook 开发，用于储存特别大的数据。Facebook 目前在使用此系统。 Cassandra 也可以在非 row-key 上实现二级索引，并且可以由多种实现方式，如 Wide Row、Super Column 和 Composite Column 三种方式。二级索引可以提供在索引列上的简单查询，但还不支持范围查询。

3.1.3 MongoDB

MongoDB 是一个基于分布式文件存储的数据库, 由 C++语言编写而成, 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。 MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。MongoDB 可以支持多种不同类型的索引，详细信息如表 5 所示：

表 5 MongoDB 支持的索引类型

序号	索引类型	特点
1	唯一索引	索引值不能重复
2	稀疏索引	索引值可以重复
3	复合索引	可以在多个字段上建立索引
4	地理空间索引	可以对空间对象进行索引

3.1.4 HugeTable

“大云”计划是中国移动研究院为打造中国移动云计算基础设施实施的关键技术，目前可实现分布式文件系统、分布式海量数据仓库、分布式计算框架、集群管理、云存储系统、弹性计算系统、并行数据挖掘工具等关键功能。其中，HugeTable 是一种高可靠，可伸缩，高性能的海量结构化信息存储系统，能够实现分布式海量数据仓库。

图 7 是 HugeTable 的系统架构，其中 Index Store 主要用来提供索引支持，基于不同的存储引擎和查询要求，HugeTable 设计了密集索引、稀疏索引和块索引三种索引方案。其中，在密集索引中，针对每条数据记录都会建立一条索引项， B+-tree 树是一种常用的密集索引结构。密集索引的主要优势是能够在索引列上提供高效的点查询和范围查询。为了减少索引空间开销，HugeTable 又提供了另外一种索引结构-稀疏索引，稀疏索引用来记录每个数据块中所包含的索引列的最大值和最小值，在查询的时候，通过将待查询值与索引项中的最大值和最小值的比较确定候选索引项。如果索引列的值分布比较均匀、且顺序存储的情况下，稀疏索引的性能较好，但是如果索引列的值随机分布，则稀疏索引效率较低。针对电信应用的具体特点，HugeTable 又提出了块索引方案。块索引的基本格式为<key, fileID,blockID>，表示在 blockID 块中包含可某个 key。与密集索引相比，块索引具有索引速度快，查询性能高和装载速度快等优点。

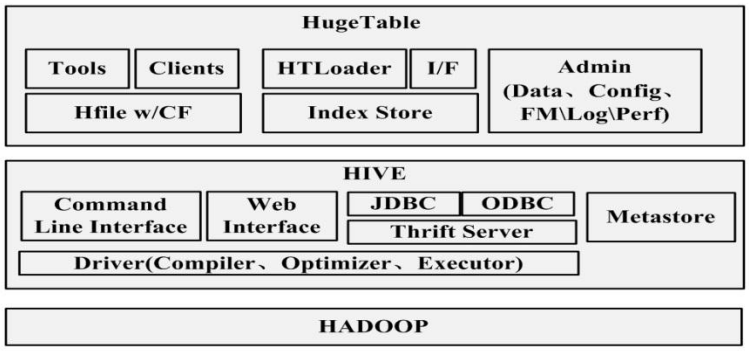

图 7 HugeTable 系统架构

针对不同索引方案的特点，HugeTable 设计了如下的查询框架，如图 8 所示：当提交一个新的查询时，HugeTable 会首先分析查询列的索引情况：如果查询列有索引，则可以直接通过索引来查找所需数据；如果查询列上没有建索引，则根据应用和数据量本身的特点来确定不通的查询方式。如果数据量比较少，则可以直接采取顺序扫描的方式来查询，如果数据量比较大，则可以利用 MapReduce 方式来并行查询，从而提高查询性能。

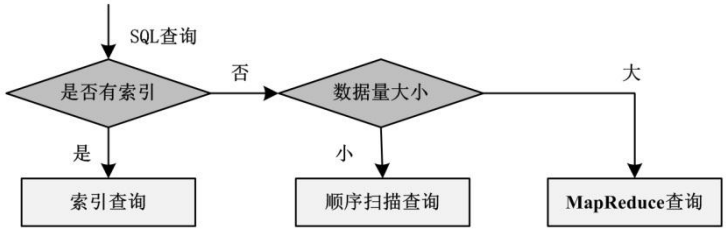

图 8 HugeTable 查询框架

3.2 海量数据处理中的索引技术

目前 Hadoop 经逐渐成为海量数据存储与处理的标准平台，通过 HDFS 可以实现海量数据高效存储、管理，通过 MapReduce 可以实现海量数据的并行查询与处理。但是由于缺乏索引技术的支持，使得某些复杂查询（如针对非 row-key 的单维或多维查询）的执行效率比较低。为了提高海量数据的复杂查询处理的效率，很多学者在已有的 Hadoop 相关技术的基础上，进行了索引技术的研究，期待基于这些索引技术，能够提高利用 MapReduce 对海量数据的查询处理效率。表 6 给出了各种索引方案的详细信息。

表 6 MapReduce 中的索引技术

方案名称	优点	缺点
HadoopDB	将 Hadoop 框架与并行数据库进行有机结合； 可以利用并行数据库提供的索引功能及高效的查询性能。	需要对 Hadoop 框架进行大量修改
Trojan Index	对 Hadoop 进行了非入侵式的优化，便于和新的 Hadoop 框架兼容；	创建索引的代价比较大
HAIL	针对每一个数据块的物理副本建立不同的索引； 索引是在数据上传的时候建立的，所以创建的代价比较小	需要修改 Hadoop 框架中的部分接

HadoopDB[27]首次尝试将 Hadoop 框架与并行数据库结合起来：Hadoop 中的 HDFS 能够实现大规模数据的分布式存储，MapReduce 可以支持大规模数据的高度并行处理；并行数据库则支持索引、视图等功能，具有丰富的查询功能。HadoopDB 将两者的优势有机地结合在了一起，大大提高了复杂查询的性能。但是为了能够与并行数据库有机结合，需要对 Hadoop 框架进行相应的修改，如果有新版本的 Hadoop 出现，则无法兼容，需要对新版本的 Hadoop 框架重新进行修改。

Hadoop++[25]则对 Hadoop Map Reduce 进行了非入侵式优化，即不需要对 Hadoop 框架本身做任何修改，这主要是通过自定义函数（UDF）来实现的，如 split、map、reduce 等。Hadoop++对 Hadoop 的优化主要包括 Trojan Index、Trojan Join 和 Trojan Layout 三个方面。其中 Trojan index 的核心是针对 HDFS 中的每一个逻辑块（logical block）创建一个块级别的索引。通过用户自定义函数，将每一个逻辑块中得数据进行重新组织，组成一个新的依次由数据、索引、Header 和 Footer 四部分构成的 block，其中 Footer 是 split 的分界符，最后一个 Footer 一定位于文件末尾。索引构建时由 MapReduce 完成排序。查询时 split 函数从文件末尾开始根据 Footer 信息解析出各个 split，itemize 函数根据搜索范围条件快速定位满足条件的内容, 这样就可以大大提高查询的性能。但是 Hadoop++创建索引的代价比较大，原因是当数据上传到 HDFS 以后，需要用一个新的 MapReduce job 来对每一个逻辑块建立索引，其开销甚至比对数据进行全表扫描的代价还要高。

为了降低创建索引的额外开销，HAIL[28]提出了一种入侵式的索引库方案（Hadoop Aggressive Indexing Library）。其主要思想是针对 HDFS 中每一个 block 的每一个物理备份分别建立不同的聚簇索引，这样每一个备份就可以在不同的属性上建立索引，从而可以支持多个属性的查询。为了降低建立索引的开销，HAIL 是在上传数据的过程中来建立索引的，主要是对原有的 HDFS client 进行修改得到 HAIL client，通过 HAIL client 来上传数据，在上传数据的过程中，对每一个副本分别建立聚簇索引，并且索引建立的过程是在内存中完成的，索引开销比较小。同时 HAIL 还对 MapReduce JobClient 进行了修改，使得在执行 MapReduce Job 之前首先对已经建立的索引进行选择，然后再选择相应的物理备份进行处理，从而就可以使用已有的索引，大大提高查询的效率。

除了针对 HDFS 和 MapReduce 的通用索引方案之外，在某些具体问题的处理中也用到了一些特定的索引技术。如李飞飞等人[29]在 MapReduce 环境下，利用基于 z-order 划分的方法来解决 KNN Join 的问题，其中在 reduce 任务中执行连接操作的时候，利用 R-tree 对局部数据进行了索引，从而提高连接的速度。Rares Vernica[30]等人利用 MapReduce 技术来解决集合相似性连接的问题，作者利用 prefix-filtering[31]技术对所有的集合建立倒排索引，然后在倒排索引的基础上进行比较，从而大大减少了比较的次数，提高了执行效率。

3.3 不同应用中的索引技术

随着云计算技术的不断发展和广泛应用，云计算和云数据管理技术在不同应用领域中都得到了尝试和应用，如海量轨迹数据管理、空间数据、文本数据以及图数据管理。本文主要对上述几种不同应用领域中的涉及到的索引相关技术进行分析总结。

3.3.1 轨迹数据

随着物联网技术、GPS 定位技术、智能手机和基于位置的服务等相关技术的快速发展和广泛应用，产生了越来越多的移动数据，传统的集中式处理技术已经无法高效处理如此大规模的移动数据，因此有部分学者开始研究基于云计算的海量移动数据处理技术。希望通过对海量轨迹数据的高效存储和处理，为智能交通提供决策支持。

文献[32]主要对海量轨迹数据的存储与检索进行了研究，作者采用 HDFS 来对轨迹数据进行存储，利用 MapReduce 提供查询。作者以线段模型来表示轨迹数据，并设计了一种静态-动态相结合的轨迹数据划分方案，实现了轨迹数据的高效存储，如图 9 所示。在划分的过程中，首先利用四叉树技术对轨迹数据在空间上进行划分，然后在每个划分区间内再根据时间维度进行划分，在时间区间的选择上是可以动态变化的，如果某个空间区域内的轨迹数据比稠密，则时间间隔可以小一些，否则时间间隔可以大一些，从而实现轨迹数据的有效划分，尽可能保证在空间、时间上相邻的轨迹数据存放在相同的分块中。每一块中的轨迹存储在 HDFS 中得一个 block 中。

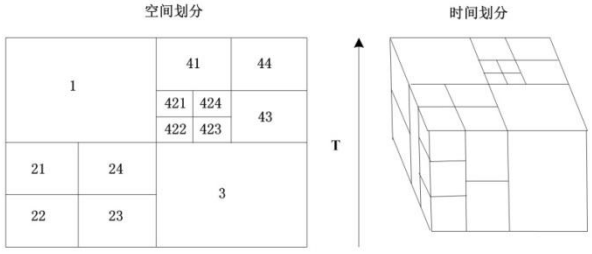

图 9 数据划分策略

为了实现轨迹数据的快速查询，作者提出了两种不同的索引方案。针对空间-时间的范围查询，作者提出了 Partition based Multilevel Index（PMI），针对轨迹查询，作者提出了 Object Inverted Index（OII）。图 10 描述了其处理框架。

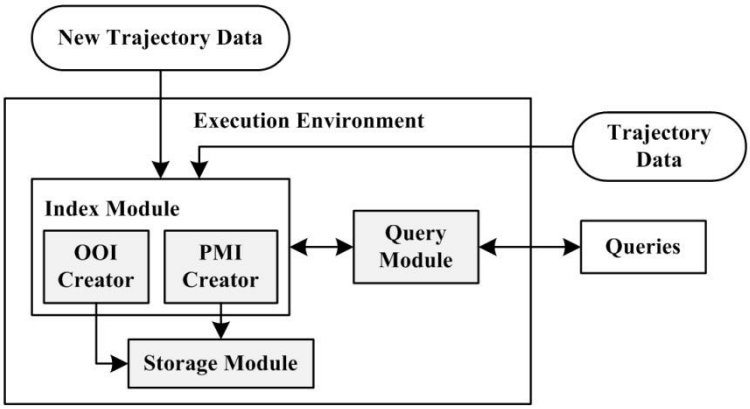

图 10 轨迹数据处理框架

PMI 主要是依据前面描述的划分方案，把处于同一划分中的轨迹线段存储在同一个数据块中，并以<PartitionID, S_k>的形式记录下来，这样在进行空间-时间范围查询的时候，就可以首先根据划分方案，计算出需要查询的轨迹可能位于那些块中，然后直接到相应的块中去查询，这样就可以大大节省查询的开销。OII 主要是为了查询某个车辆的轨迹而设计的，通过搜集某个车辆的所有轨迹线段，并以<OID, { PartitionID, T}>的形式记录下来，实际上相当于是建立了一个倒排索引表。这样当需要查询某个特定车辆的轨迹的时候，就可以直接利用 OII 找到相应的车辆，从中即可查出其对应的轨迹。

为了提高大规模轨迹查询的效率，文献[33]在 HBase 的基础上提出了一种可扩展的轨迹索引方案。根据轨迹数据的特点，作者主要设计了两种表结构来实现轨迹数据的存储和索引，主表（main table）主要是用来存储原始轨迹数据，包括 GPS 坐标及其他轨迹相关数据；轨迹索引表（trail indices table）主要是对空间维度进行不同粒度的划分，然后在不同的粒度上对空间点分别建立索引，这样查询的时候就可以从粗粒度开始进行过滤，大大提高查询效率。

表 7 主表

Row key	Timestamp	Column <*UserData*>:<*trail_id*>
x1_y1	*t(2)*	*trailA; trailD; trailE; trailF*
	t(1)	*trailA; trailD*
X2_y2	*t(1)*	*trailB; trailC*
X3_y3	*t(3)*	*trailG*

表 8 轨迹索引表

Row key	Column <*L*>:<*trail_id*>	Column <*L+1*>:<*trail_id*>
$I_{x1}(L)$, $I_{y1}(L)$	*trailA; trailD; trailE; trailF*	
$I_{x2}(L)$, $I_{y2}(L)$	*trailB; trailC*	
$I_{x3}(L+1)$, $I_{y3}(L+1)$		*trailF, trailG*

文献[34]结合浮动车数据的实际特点，设计了基于 Bigtable 的存储模型，并分别针对路网数据和浮动车数据分别建立了相应的索引，以提高查询处理的效率。存储模型如图 11 和图 12 所示：

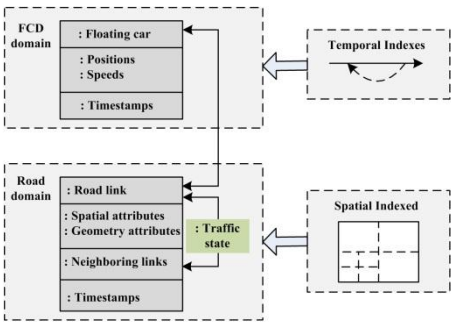

图 11 概念视图

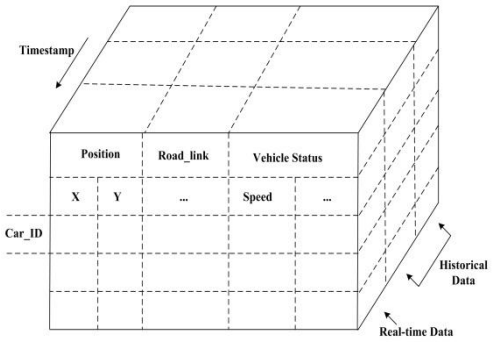

图 12 浮动车数据表结构

为了查询出租车在某段时间内的运行轨迹，作者针对浮动车数据建立了时间索引，实际上是利用时间戳和车辆 ID 组合形成新的 row-key，建立了二级索引。同时作者对路网数据建立了四叉树索引，方便进行地图匹配和车速计算。地图匹配和车速计算是浮动车数据处理常见任务，可以为智能交通提供决策支持。为了提高地图匹配和车速计算的效率，作者利用 MapRduce 技术进行了优化。

作者提出的存储模型和索引方案，虽然可以提高某些查询的效率，但是对其他一些复杂查询，仍然无法实现较好的支持，如多维度的范围查询（某时间段内、某区域范围内的车辆）。所以，设计一种能支持多种不同类型查询的存储与索引方案仍然是一件极具挑战性的工作

3.3.2 空间数据

随着空间数据对象的不断增加，如何在海量空间数据对象中实现高效、快速的空间查询成为一个日益重要的问题。文献[35]重点描述了海量空间数据库中的空间查询问题。作者首先分析了传统的基于 R-tree 的索引结构的一些缺点和不足之处，然后基于云计算技术提出了一个新的可扩展的解决方案。 基本思路如图 13 所示：方案主要包括三个阶段，第一阶段是数据划分。为了找到一个比较好的数据划分方案，作者通过采样的方法，得到相应的数据划分方案，并以此作为整体数据的划分方案。第二阶段是生成 R-tree 的阶段，根据第一阶段得到的划分方案，对所有数据在空间上划分为互不相交的若干部分，对于每一部分的数据分别建立 R-tree 索引。由于各部分数据是不重叠的，索引的建立可以采用并行的方式来进行，这样可以大大提高索引的构建速度。作者采用 MapReduce 的方式来进行索引的建立，Map 阶段的主要任务是根据前一阶段的划分方案把每一个空间对象划分到相应的分区中，Reduce 阶段分别对每一个分区中的数据建立 R-tree。第三个阶段的任务是对第二阶段所产生的各个子 R-tree 进行合并，生成一颗大的 R-tree。查询的时候则首先根据整体的索引树找出所查找对象所在的子 R-tree，然后在分别到对应的子 R-tree 上去进行查询，最后在对查找到的结果进行合并。

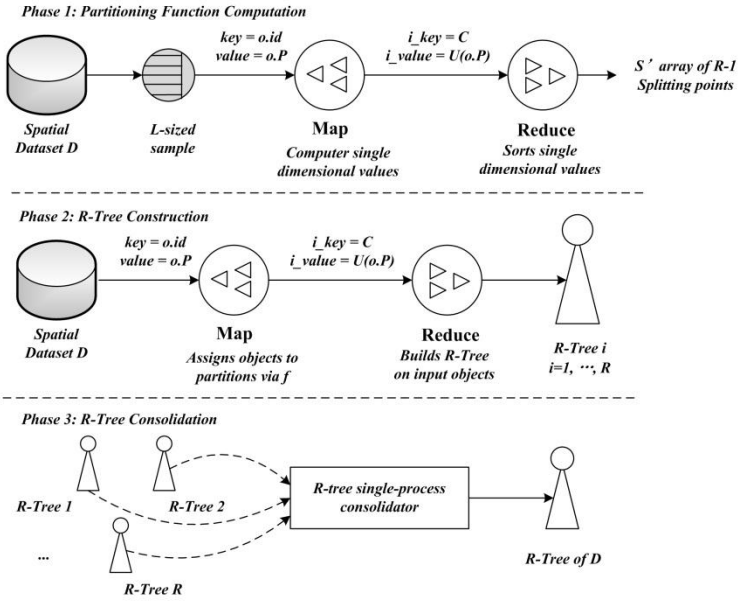

图 13 索引过程

文献[36]针对海量空间数据提出了一种基于 R^+-tree 和 key-value 健值生成规则的索引结构 KR^+-index，其基本结构如图 14 所示：

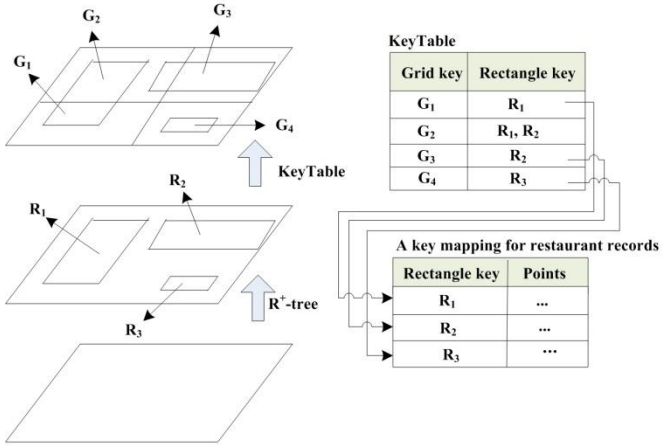

图 14 KR^+-index 基本结构

KR^+-index 首先利用 R^+-tree 对数据进行划分，把数据划分成若干个互不相交的矩形，然后再利用网格对空间进行划分，并利用 Hilbert-curve 对这些网格单元进行编码，根据网格单元和矩形的交叉情况，记录下网格编号与矩形单元的对应关系。查询的时候，首先根据查询范围在 KeyTable 中找出数据所在的矩形单元，然后再根据所找到的矩形单元去查找相应的数据。

3.3.3 图数据

随着社交网络分析、语义 Web 分析、生物信息网络分析等新兴应用的快速增长，对亿万个顶点级别大规模图的处理能力的需求愈加迫切，于戈等人[37]结合云计算的特点，从图数据管理与图数据处理机制两个方面，综述了云计算环境下进行大规模图数据处理的关键问题，并对大规模图数据的索引技术进行了总结。文中首先对目前云计算环境下的通用索引技

术进行了简单分析，并指出这些索引技术虽然不是专门针对图数据进行设计的，但是这些索引技术大都可以被大规模图数据管理所借鉴。文中重点对两种分布式图数据库中所采用的索引技术进行了分析。

Neo4j 的索引分为两类：数据库本身就是一个树形结构的索引，可用于提高查询效率，此外，还可以使用独立的 Lucene【脚注】索引，提供全文索引和索引命中率排序功能。Neo4j 可以对图顶点和边分别建立索引，通过对索引的缓存，可进一步加快查找速度。索引的维护操作（如删除、更新）则必须在事务管理机制下进行。对于更新，必须先删除旧的索引值，然后才能添加新索引值，代价较高．Trinity 数据库提供 Trie 树和 Hash 两种索引结构来访问图顶点、边的名字以及相关的其它信息，可减少有公共前缀的字符串的匹配次数。

4 我们的工作

在对云数据管理系统中现有索引技术进行深入调研分析的基础上，我们团队针对云计算环境下海量物联网数据的多维索引也做了一些研究工作，如“An Efficient Index for Massive IOT Data in Cloud Environment”[38]。

随着 RFID、GPS 等技术的发展，物联网得到了迅速普及和广泛应用。在物联网环境下，往往会有数以十万、百万计的传感器周期性地产生数据，物联网数据海量性使得传统的关系型数据库在扩展性方面遇到了瓶颈，并且无法支持每秒数万、甚至数十万的并发操作。而云数据管理系统天然具有高扩展性、可用性以及容错性，是海量物联网数据管理的有效方案。但是正如前面章节所述，云数据管理系统也有自己的不足，即只能在 row-key 上提供高效的点查询和范围查询，对于非 row-key 的查询则需要全表扫描，虽然可以利用 MapReduce 技术来提高查询的效率，但是对于选择率比较低的查询来说，性能还是比较差。而物联网数据往往具有多维特性，除了具有时间、空间属性之外，还有很多其他维度的信息。并且对物联网数据的查询也大都是基于时空的多维查询，因此在物联网应用环境中，除了要满足单个维度的快速查询之外，还需要提供高效的多维查询。[38]的主要工作就是根据物联网数据的特点，在云计算环境下设计一种同时支持高效更新和快速多维查询的索引方案。主要创新点有两个，一是根据设计了一种多层索引框架，二是根据网联网数据的时空特性，设计了一种动态的数据划分方案。并且将索引方案跟 HBase 相结合，开发了原型系统。

为了同时支持高效的频繁更新和快速的多维查询，我们主要采取了两个措施：一是对当前数据和历史数据在不同的粒度级别上进行索引，在数据写入的过程中，尽可能减少索引更新的次数，从而降低索引维护的代价；二是尽量把索引和数据分开，减少索引创建对系统写入性能的影响。图 15 和图 16 分别描述了系统架构和多层索引框架。

图 15 展示了索引的框架，主要包括三个层次，其中时间段索引和子空间索引主要针对当前数据进行索引，网格索引主要针对历史数据进行索引。由于物联网数据具有时空连续性，并且不同时刻数据在空间上的分布往往会发生变化，所以我们把时间维度和空间维度分开进行考虑。首先在时间维度上分成若干个时间段，针对每个时间段内的数据，在二维空间中进行划分，划分成若干个子空间，每个子空间内的数据存储到 HBase 中的同一个 region 中，这样就能够保证在时间和空间上临近的数据尽量存储在相同的区域中，减少查询过程中需要扫描的 region 的数量，从而提高查询效率。

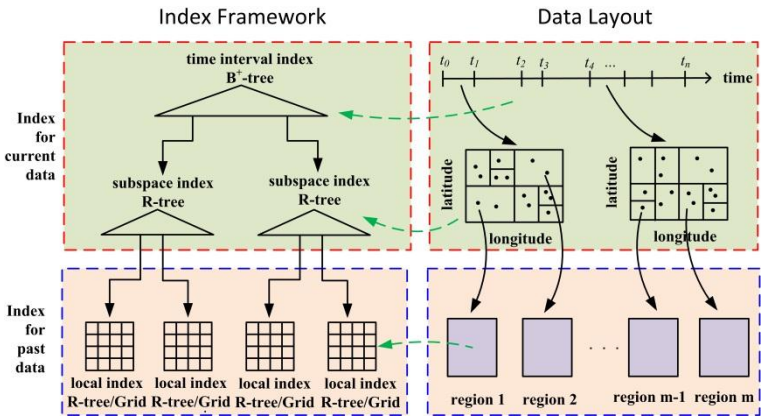

图 15 多层索引框架

在索引的时候，我们在时间维度上把数据分成当前时间段的数据和历史时段的数据。对于当前时段的数据，仅仅对其所在的时间段和所在的子空间进行索引，而不对数据记录本身进行索引，这样在数据插入的时候就可以大大减少索引更新的次数。其中时间段可以采用 B^+-tree 进行索引，由于每个子空间都是互不相交的矩形区域，所以采用 R-tree 进行索引。当前时间段结束以后，数据开始从新的时间段插入，历史时段的数据不再发生改变，此时可以批量地对历史时段中每个 region 内部的数据建立记录级别的索引，可以采用 R-tree 索引或者网格索引。该索引方案索引更新维护的代价比较低，对数据插入性能的影响比较小，从而保证系统能够支持大规模的频繁更新。

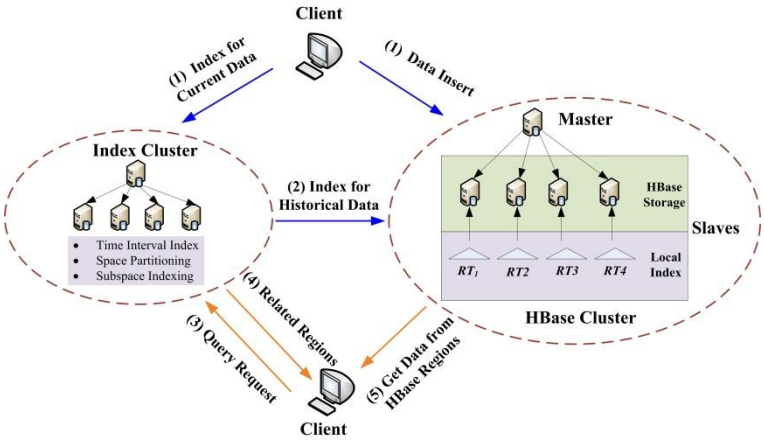

图 16 系统架构

图 16 显示了系统的整体架构，主要包括客户端、索引集群和 HBase 集群。其中客户端主要负责发起数据的插入和查询；HBase 集群负责数据的存储和历史数据的索引；Index 集群主要负责当前数据索引的创建和维护。当新数据来临的时候，会同时发送到 HBase 集群和 Index 集群，通过 HBase 集群完成数据的存储，通过 Index 集群，对当前数据在较粗粒度级别(时间段和空间区域)上进行索引，这样就可以实现索引的创建维护与数据的存储相分离，索引的创建不会对插入性能有太大的影响；索引集群主要针对当前数据在时间段和空间区域上进行粗粒度的索引，当前时间段结束以后，已经插入的数据不再发生改变，索引集群发出请求，针对历史数据在 HBase 集群中建立记录级别的索引；当进行查询的时候，首先通过索引集群对数据所在的 region 进行过滤，获得数据所在的 region；然后再根据得到的 region

信息到 HBase 中进行查询。

最后我们结合 HBase 开发了原型系统，并做了大量实验，实验结果表明我们所提出了索引方案既能支持高效的多维查询，又能保证较高的系统吞吐率，达到了预期目的。

5 未来工作展望及挑战

虽然目前针对云数据管理中的索引技术已经有了很多研究工作，但是目前的研究成果还不是十分成熟，还未能得到广泛的应用。并且随着 Web 2.0、物联网、移动互联网的快速发展和广泛应用，数据呈现爆炸式增长，人类已经进入了大数据时代。云计算是解决大数据的一种有效方式，而随着数据量的不断增长以及数据复杂性的不断增加，在云计算环境下针对大数据的索引技术也面临着一系列新的挑战。

5.1 复杂数据类型的索引

如前所述，随着社交网络、物联网技术、移动互联网、智能手机等相关技术的快速发展，一些复杂类型的数据量在以指数级增长，给传统的数据管理方法及索引技术带了巨大挑战，如图数据、轨迹数据、序列数据、空间数据等。针对大规模图数据而言，虽然基于云计算环境下的并行处理机制可以提高其管理和处理效率，但是由于图数据本身的复杂性、特殊性，使得现有的云计算处理方式无法很好地适应大规模图数据的管理。在大规模图数据处理中，只有很少的处理应用了云计算环境下的索引技术，如最短路径计算等[37]，大部分的图处理问题还没有考虑到索引的应用，有待于进一步深入研究。轨迹数据也是越来越重要的一种数据类型，主要是得益于智能手机的普遍应用以及移动互联网

的快速发展，如基于位置的服务（LBS）、基于位置的社交网络（LBSN）、FourSquare、车辆的 GPS 数据等。虽然针对轨迹数据的索引研究已经有很多相对比较成熟的技术，但是在现有的应用环境下，数据量已远非过去的应用所能比，再加上轨迹数据往往更新比较频繁，同时也具有多维特性，使得已有的轨迹索引技术无法在云计算环境下有效的管理海量轨迹数据。

5.2 复杂查询的索引

复杂查询主要包括多维查询、最近邻查询（KNN）、Top-K 查询、Join 以及 KNN-Join 等。上述复杂查询类型在数据规模比较小的情况下，都有比较成熟的索引技术和查询方案，可以实现高效查询。但是，在大数据情况下，原有的索引技术和查询处理方法已经无法满足性能要求。并且目前的云计算技术如 MapReduce 还不能很好地支持多维查询、连接查询等复杂操作。因此，在云计算环境下，设计特定的索引方案，从而提高针对大数据的、复杂查询的处理效率，是一个重要的研究问题。

5.3 通用索引与专用索引

我们认为，传统的一些通用的、成熟的基于磁盘的索引技术如 R-tree、B-tree 等，在大数据场景下无法有效工作。当数据量特别大，尤其是数据更新特别频繁时，传统索引的更新维护和代价就会变得特别高，甚至超过索引本身带来的优势。通用索引可以用来支持一些常见的、简单的数据查询，同时通用索引本身也不能够太复杂，必须简单。而针对一些相对比较复杂的查询处理任务，则必须根据数据本身的特点以及数据处理任务的具体要求设计专用的索引结构，这种类型的索引结构可以相对复杂一些。

5.4 索引的动态增加与删除

在实际应用当中，业务类型经常会发生改变，查询类型也就需要进行相应的改变。并且，实际应用中的数据往往有很多属性，几十个甚至上百个属性，事先不可能对所有的属性都建立相应的索引，这样代价大。因此，在云计算环境下，必须设计一些巧妙的索引方案，使得索引能够支持动态的增加与删除，并且索引更新和维护的代价又不能够太大，必须在可接受的范围内。

5.5 基于索引的查询优化

云计算环境下，基于 MapReduce 的查询优化技术已经有一些相关的研究工作[39]，如 MapReduce 在多核硬件与 GPU 上的改进[40]、基于调度技术的 MapReduce 性能优化[41]，还有一些工作对现有的 MapReduce 框架进行了修改，从而提高查询效率[42]。也有一些工作基于索引进行查询优化，如 CCIndex[13]、Hadoop++[27]。但是目前的查询优化技术还比较简单，无法很好地满足实际应用的需求，因此在这方面还需要进一步深入研究，如基于代价的查询优化等。

5.6 其他

除了上述几点挑战性工作之外，还有一些其他的工作可以探索。回顾数据处理的历史我们可以发现，以前在数据规模比较小的情况下，我们的计算模式是以计算为中心，把数据移动到计算单元，供计算单元所使用。因为对于小规模数据来讲，计算开销占主导地位。然而在大数据场景下，相对于计算开销而言，数据传输开销则占据了主导地位，所以现在就产生了以数据为中心的云计算模式，把计算单元推送到数据单元去执行。基于这种模式的启发，我们是否可以把索引与数据单元结合在一起，索引可以看成是一种特殊的计算，让每个数据单元都具有索引功能，并且可以自动管理和演化。当然，这只是一种设想，期待未来会有一定的突破。

6 结束语

随着数据的爆炸式增长，我们已经进入了大数据时代。由于云计算技术本身的一些优势如高扩展性、可用性、容错性等，使得云计算技术得到了广泛应用，因此云计算也就成了大数据存储、管理与处理的一种有效方式。然而云计算本身也存在一些不足之处，如缺乏对索引的支持就是其中之一，这限制了云计算在很多方面的应用。本文对云数据管理的索引技术的相关工作进行了深入研究，并作了分析、对比和总结，指出了其中各自的优点及不足；同时对我们在物联网数据多维索引方面的研究工作进行了介绍；最后指出了在云计算环境下针对大数据索引技术的若干挑战性问题，并进行了简单分析。该问题的研究还处于起步阶段，还没有比较成熟的、可以进行大规模实际应用的成果和方案，因此，具有重要的研究价值和实际意义。

参 考 文 献

[1] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. The google file system//Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York , USA, 2003: 29-43

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat et al. Bigtable: A Distributed Storage System for Structured Data// Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. Seattle, WA, USA, 2006: 205-218

[3] Jeffrey Dean, Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters//Proceedings of the 5th USENIX Symposium on Operating Systems Design and

Implementation. San Francisco, California, USA, 2004: 137-150
[4] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava et al. PNUTS: Yahoo!'s hosted data serving platform.// Proceedings of the PVLDB. Auckland, New Zealand, 2008: 1277-1288
[5] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani et al. Dynamo: amazon's highly available key-value store// Proceedings of the 21st ACM Symposium on Operating Systems Principles. Stevenson, Washington, USA, 2007: 205–220
[6] Sai Wu, Dawei Jiang, Beng Chin Ooi et al. Efficient B-tree Based Indexing for Cloud Data Processing. PVLDB, 2010, 3(1):1207–1218
[7] Jinbao Wang, Sai Wu, Hong Gao et al. Indexing multi-dimensional data in a cloud system//Proceedings of the ACM SIGMOD/PODS Conference. Indianapolis, USA, 2010: 591-602
[8] Andreas Papadopoulos, Dimitrios Katsaros. A-Tree: Distributed Indexing of Multi-dimensional Data for Cloud Computing Environments//Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science. Athens, Greece, 2011
[9] Xiangyu Zhang, Jing Ai, Zhongyuan Wang et al. An efficient multidimensional index for cloud data management//Proceeding of the First International Workshop on Cloud Data Management. Hong Kong, China, 2009: 17-24
[10] LinLin Ding, Baiyou Qiao, Guoren Wang et al. An Efficient Quad-Tree Based Index Structure for Cloud Data Management//Proceedings of the 12th International Conference on Web-Age Information Management. Wuhan, China, 2010: 238-250
[11] Huang Bin, Peng Yu-Xing. An efficient two-level bitmap index for cloud data management//Proceedings of the 3rd IEEE International Conference on Communication Software and Networks. Xi'an, China, 2011: 509-513
[12] Parag Agrawal, Adam Silberstein, Brian F. Cooper et al. Asynchronous view maintenance for VLSD databases.//Proceeding of the ACM SIGMOD/PODS Conference. Providence , USA, 2009: 179-192.
[13] Yongqiang Zou, Jia Liu, Shicai Wang et al. CCIndex: a Complemental Clustering Index on Distributed Ordered Tables for Multi-dimensional Range Queries//Proceedings of the 7th IFIP International Conference on Network and Parallel Computing. Zhengzhou, China, 2010: 247-261
[14] Marcos Kawazoe Aguilera, Wojciech M. Golab, Mehul A. Shah. A practical scalable distributed B-tree. PVLDB, 2008, 1(1): 598-609
[15] Shoji Nishimura, Sudipto Das, Divyakant Agrawal et al. MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services//Proceedings of the 11th International Conference on Mobile Data Management. Kansas City, Missouri, USA, 2011: 7-16
[17] Sai Wu, Kun-Lung Wu. An indexing framework for efficient retrieval on the cloud. IEEE Data Eng. Bull. 2009, 32(1): 75-82
[18] H. V. Jagadish , Beng Chin Ooi , Quang Hieu Vu. 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
[19] Ion Stoica, Robert Morris, David R. Karger et al. Chord: A scalable peer-to-peer lookup service for internet applications// Proceedings of the ACM SIGCOMM. California, USA, 2001: 149-160
[20] Jon Louis Bentley. Multidimensional Binary Search Trees in Database Applications. IEEE Transactions on Software Engineering, 1979, 5(4): 333-340

[21] Hailong Cai, Ping Ge, Jun Wang. Applications of Bloom Filters in Peer-to-peer Systems: Issues and Questions//Proceedings of the International Conference on Networking, Architecture, and Storage. Chongqing, China, 2008: 97-103
[22] Morton, G. M. A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. Technical Report, Ottawa, Canada: IBM Ltd, 1966
[23] Meng Bi-Ping, Wang Teng-Jiao, Li Hong-Yan, Yang Dong-Qing. Regional Bitmap Index: A Secondary Index for Data Management in Cloud Computing Environment. Chinese Journal of Computers, 2012, 35(11): 2306-2316
（孟必平，王腾蛟，李红燕，杨冬青. 分片位图索引：一种适用于云数据管理的辅助索引机制. 计算机学报, 2012, 35（11）：2306-2316）
[24] Sylvia Ratnasamy, Paul Francis, Mark Handley et al. A scalable content-addressable network//Proceedings of the ACM SIGCOMM. California, USA, 2001: 161-172
[25] Jens Dittrich, Jorge-Arnulfo Quian é-Ruiz, Alekh Jindal et al. Hadoop++: Making a Yellow Elephant Run Like a Cheetah. PVLDB, 2010, 3(1): 518-529
[26] Zhou Da, Qian Ling, Guo Lei-Tao, Qi Ji. HugeTable: Telecom Oriented Data Warehouse. Computer Science, 2011, 38(8) Supp: 186-190
（周大，钱岭，郭磊涛，齐骥. HugeTable: 一种面向电信行业的云数据仓库. 计算机科学，2011，38（8）增刊：186-190）
[27]Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi et al. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. PVLDB, 2009, 2(1): 922-933.
[28] Jens Dittrich, Jorge-Arnulfo Quian é-Ruiz, Stefan Richter et al. Only Aggressive Elephants are Fast Elephants. PVLDB, 2012, 5(11): 1591-1602
[29] Chi Zhang, Feifei Li, Jeffrey Jestes. Efficient parallel kNN joins for large data in MapReduce//Proceedings of the 15th International Conference on Extending Database Technology. Berlin, Germany, 2012: 38-49
[30] Rares Vernica, Michael J. Carey, Chen Li. Efficient parallel set-similarity joins using MapReduce//Proceedings of the ACM SIGMOD/PODS Conference. Indianapolis, USA, 2010: 495-506
[31] Surajit Chaudhuri, Venkatesh Ganti, Raghav Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning//Proceedings of the 22nd International Conference on Data Engineering. Atlanta, GA, USA, 2006: 5
[32] Qiang Ma, Bin Yang, Weining Qian, Aoying Zhou. Query Processing of Massive Trajectory Data Based on MapReduce // Proceeding of the First International Workshop on Cloud Data Management. Hong Kong, China, 2009: 9-16.
[33] Shih-Tsun Chu, Chao-Chun Yeh, Chun-Lung Huang. A Cloud-Based Trajectory Index Scheme//Proceedings of the IEEE International Conference on E-Business Engineering. Macau, China, 2009: 602-607
[34] Li Qing-quan, Zhang Tong, Yu Yang. Using cloud computing to process intensive floating car data for urban traffic surveillance. INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE, 2011, 25 (8): 1303-1322
[35] Ariel Cary, Zhengguo Sun, Vagelis Hristidis et al. Experiences on Processing Spatial Data with MapReduce// Proceedings of the 21st International Conference on Scientific and Statistical Database Management. New Orleans, LA, USA, 2009: 302-319

[36] Ya-Ting Hsu, Yi-Chin Pan, Ling-Yin Wei et al. Key Formulation Schemes for Spatial Index in Cloud Data Managements//Proceedings of the 13th IEEE Conference on Mobile Data Management. Bangalore, India, 2012: 21-26
[37] Yu Ge, Gu Yu, Bao Yu-Bin, Wang Zhi-Gang. Large Scale Graph Data Processing on Cloud Computing Environment. Chinese Journal of Computers, 2011, 34(10): 1753-1767
（于戈，谷峪，鲍玉斌，王志刚. 云计算环境下的大规模图数据处理技术. 计算机学报，2011，34（10）：1753—1767)
[38] Ma You-Zhong, Rao Jia, Hu Wei-Song, Meng Xiao-Feng et al. An Efficient Index for Massive IOT Data in Cloud Environment//Proceedings of the 21st ACM Conference on Information and Knowledge Management. Maui, USA, 2012: 2129-2133
[39] Qin Xiong-Pai, Wang Hui-Ju, Du Xiao-Yong et al. Big Data Analysis-Competition between RDBMS and MapReduce, Their Inter-Fusing and Symbiosis. Journal of Software, 2012, 23(1): 32-45
（覃雄派,王会举,杜小勇 等. 大数据分析--RDBMS 与 MapReduce 的竞争与共生. 软件学报，2012， 23（1）：32-45）
[40] Bingsheng He, Wenbin Fang, Qiong Luo et al. Mars: a MapReduce framework on graphics processors//Proceedings of the 7th International Conference on Parallel Architectures and Compilation Techniques. Toronto, CANADA, 2008: 260-269
[41] Thomas Sandholm, Kevin Lai. MapReduce optimization using regulated dynamic prioritization//Proceedings of the SIGMETRICS/Performance. Seattle, WA, USA, 2009: 299-310
[42] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao et al. Map-reduce-merge: simplified relational data processing on large clusters//Proceeding of the ACM SIGMOD/PODS Conference. Beijing, China, 2007: 1029-1040