·人工智能与模式识别· 文章编号：1000-3428(2021)03-0062-09 文献标志码：A 中图分类号：TP311

面向轨迹流数据的索引构建与存储方法研究

蔡瑞初[1]，林峰极[1]，郝志峰[1,2]，王 立[3]，温 雯[1]
（1.广东工业大学 计算机学院，广州 510006；2.佛山科学技术学院 数学与大数据学院，广东 佛山 528000；
3.依图网络科技有限公司 新加坡研发部，新加坡 018960）

摘 要：移动社交网络等基于定位服务应用的快速发展导致时空数据流规模呈爆炸式增长，要求底层数据存储系统支持高吞吐量轨迹数据的插入以及空间和时间约束下的低延迟查询，而现有HBase等数据存储方案因索引更新开销过高无法满足该需求。针对时空数据流的应用特性，提出一种数据流内存索引及存储方法。根据键值和时间范围对历史与增量数据元组进行物理分区，将其以模板B+树的形式写入内存并构建索引以增强快速写入和查询能力，同时对数据进行压缩存储提升索引效率。在此基础上，采用多级索引根据数据分区将复杂查询分解为可独立处理的子查询。实验结果表明，与传统HBase、WaterWheel等方法相比，该方法在不同数据插入和查询条件下的数据存储性能与查询效率更优。

关键词：轨迹流数据；数据分区；存储；多级索引；Bloom过滤器

开放科学（资源服务）标志码（OSID）：

中文引用格式：蔡瑞初，林峰极，郝志峰，等. 面向轨迹流数据的索引构建与存储方法研究[J]. 计算机工程，2021，47(3)：62-70.

英文引用格式：CAI Ruichu，LIN Fengji，HAO Zhifeng，et al. Research on index construction and storage method for trajectory stream data[J]. Computer Engineering，2021，47(3)：62-70.

Research on Index Construction and Storage Method for Trajectory Stream Data

CAI Ruichu[1]，LIN Fengji[1]，HAO Zhifeng[1,2]，WANG Li[3]，WEN Wen[1]
（1.School of Computer，Guangdong University of Technology，Guangzhou 510006，China；
2.School of Mathematics and Big Data，Foshan University，Foshan，Guangdong 528000，China；
3.Singapore Research and Development Department，Yitu Network Technology Co.，Ltd.，Singapore 018960，Singapore）

【Abstract】The rapid development of location-based services and applications such as mobile social networks leads to the explosive growth of the scale of spatiotemporal data stream，which requires the underlying data storage system to support high-throughput trajectory data insertion and low-latency query under space and time constraints.However，the existing data storage schemes such as HBase fail to meet the requirements due to the high index update overhead. According to the application characteristics of spatiotemporal data stream，this paper proposes a data stream memory index and storage method. According to the key value and time range，the tuple of historical and incremental data is physically partitioned，which is written into memory in the form of template B+ tree，and an index is built to enhance the ability of fast writing and query. At the same time，the data is compressed to improve the index efficiency.On this basis，the multi-level index is adopted to decompose the complex query into sub-queries that can be independently processed according to the data partition. Experimental results show that under different data insertion and query conditions，the proposed method outperforms traditional HBase，WaterWheel and other methods in terms of data storage performance and query efficiency.

【Key words】trajectory stream data；data partition；storage；multilevel index；Bloom filter

DOI：10. 19678/j. issn. 1000-3428. 0057108

基金项目：国家自然科学基金（61100148，61876043）；国家自然科学基金-广东省联合基金（U1501254）；广东省自然科学基金（2014A030306004，2014A030308008）；广东省特支计划（2015TQ01X140）；广州市科技计划（201902010058）；广州市珠江科技新星专项（201610010101）。

作者简介：蔡瑞初（1983—），男，教授、博士，主研方向为数据挖掘、高性能计算；林峰极（通信作者），硕士研究生；郝志峰，教授、博士；王 立，高级工程师、博士；温 雯，教授、博士。

收稿日期：2020-01-03 **修回日期：**2020-03-02 **E-mail：**linfengji94@163.com

计算机工程 www.ecice06.com

0 概述

随着信息处理技术的快速发展,人们生活各方面产生的数据量呈现几何级增长,对分布式计算与数据存储提出更高要求。在企业大数据应用需求推动下,Hadoop、Storm和Flink等分布式计算框架相继出现并显著加快了数据处理速度,其中Storm是一种分布式流式计算处理框架,可使任务分布到集群中进行。Google公司的GFS、HDFS以及基于CEPH的CephFS等分布式存储系统支持文件及对象的快速读写,可存储大规模数据。key-value模型[1]与以Redis为代表的NoSQL型数据库是目前使用广泛的存储模型。

在移动社交网络和基于位置的服务领域,每秒都会有大量轨迹数据产生。为便于分析与管理,数据存储系统需在存储高速轨迹数据流的同时支持低延迟的时间范围查询。例如,在以每秒约百万个元组的速度记录实时GPS数据的同时,对最近5 min内获取的某个地理区域中全部GPS数据进行交互式查询。然而现有的HBase等数据存储方案无法同时实现数据的高速插入与低延迟时间范围查询。此外,对大量数据进行低延迟时间范围查询需在时空列上创建索引,且元组插入与索引更新结合后产生大量时间开销,导致高吞吐量的插入无法实现。同时,由于众多场景要求数据元组在其到达时立即可见,因此不能采用基于批处理的插入来降低索引更新成本。

为实现海量流数据的实时存储与高效查询,本文提出一种分布式数据处理方法。针对高吞吐量流数据,构建具有对应拓扑结构的分布式集群模型,采用数据分区模式和基于内存索引的压缩存储方法,降低底层文件系统负载压力并提高数据插入效率,通过多级索引机制提升复杂查询的分解与访问效率,减少无关数据的查询处理,同时构建完整的分布式存储系统,以支持join和聚集函数等数据库常用复杂查询模式。

1 相关背景

集群架构、索引结构、机器性能以及事务支持等因素均会影响流数据的写入速率,目前HBase[2]、Dynamo[3]、BigTable[4]、CLAIMS[5]和Druid[6]等数据存储方案主要通过分布式集群来存储和管理大量数据。BigTable及其开源实现HBase将数据组织为分布式多维排序映射,并可提供高效的可扩展查询,但以其为代表的键值在时间范围上对数据库查询能力较差。Druid通过数据预聚合和倒排索引实现快速查询与实时分析,可用于海量事件流数据的存储及分析。然而Druid和BTrDb[7]等时间序列数据库在时间属性外的第二维索引能力较差。

索引是一种常用技术,在高频范围属性上创建索引可显著提升查询性能。然而以高速率插入元组时,会造成索引维护开销太大。目前批量加载/插入方法是一次插入一批,并非单独插入各元组来分摊开销。当前关于索引技术的研究主要集中于哈希(Hash)函数、LSM树和B+树。例如,文献[8]提出一种分布式批量插入方法,对跨分区的负载平衡进行优化。然而,这些技术并不适用于数据元组在写入后要求立即可见的情况。LSM-tree[9]及其变形[10-11]在不进行批处理的情况下可提高插入性能,因此被广泛应用于HBase、MongoDB、Cassandra[12]和InfluxDB等多种数据库管理系统。文献[13]提出cLSM-tree对LSM树索引的并发机制进行优化,可在服务器CPU多核环境下实现高扩展性,其关键思想是将B+树维持在两层或更多层中,且较高层的节点保存在容量较小的内存中。然而由于上层数据达到一定数量后需与低层数据合并,会造成大量合并的开销,因此导致LSM树的插入性能不高。

对数据库查询日志等历史数据及记录的利用也是当前的研究热点。若数据服务系统待处理的查询请求与历史查询日志M特征相同,则可通过处理M来更新计算数据索引的值;若特征不相同,则可先预测查询负载情况,再利用查询负载对当前索引结构进行调优[14-15]。文献[16]针对日志数据设计与实现高效的并发数据,将其写入系统流水以提高数据加载性能,并允许应用程序访问加载中的数据。文献[17]结合机器学习提出学习索引结构,利用机器学习对数据建模以替代传统索引。然而查询日志和建模需要批量处理数据以及对历史数据的大规模分析,在本文实时数据场景下,采用索引优化结构无法实现实时可见,不能有效进行索引调优。

支持高吞吐量和实时查询的WaterWheel模型[18]只能在单一的查询请求下,对计算机资源实时进行最优化分配。而在分布式流式处理任务中,常出现高并发度的查询请求,会根据系统当前不同环节的资源使用与任务执行情况,对已处理完毕且处于空闲状态的任务推送下一个请求,以保证系统不同环节始终处于任务负载状态。WaterWheel是通过串行化执行每次查询请求,返回结果后再处理下一个请求,导致集群内部分节点在完成子查询请求后到下次查询请求被分配到该节点之前,一直处于查询空闲状态。

从总体来看,现有分布式存储系统具有支持流数据处理的能力,却无法提供较好的流数据高速插入与低延时时间范围查询性能。针对该问题,本文提出一种流式数据存储与查询模型Tars,在数据插入时进行内存索引和文件压缩存储,并在查询时进行查询分解以及构建多级索引。在系统模型整体搭建方面,本文基于流数据处理框架Storm构建包含调

计算机工程 www.ecice.com

度层与服务层组件的拓扑结构，调度层负责源数据的划分转发与查询的拆解分配，服务层负责数据的内存索引和子查询的任务执行。在系统模型数据存储方面，数据在内存索引中基于template B+树[19]构建索引结构，并在超过阈值后将其分组压缩存储到CephFS中（Ceph[20]是一种能自动均衡和恢复且可扩展的高性能分布式存储系统，其提供了文件系统服务CephFS）。在系统模型查询调度方面，本文基于数据范围分区划分模式对查询进行分解，通过构建多级索引，只读取符合复杂查询条件的数据文件，以保证提供高效查询。

2 流式数据存储与查询模型的基本原理

基于位置信息的流数据要实现数据的实时性和完整性，需要尽可能降低实时写入海量数据时查询数据服务带来的性能影响，并提高模型查询重复数据时的优化能力。本文主要从以下方面优化模型的存储与查询能力：

1）提高模型并发读写能力。根据数据分区划分模式，接收不断流入的数据后，将其转发到不同机器与组件进行内存索引；查询数据时，将查询切分为子查询，并将请求分发到不同索引组件和文件系统。

2）提高数据索引与存储能力。写入数据在内存中被组织为索引模式，并在达到阈值后进行压缩存储到分布式文件系统。

3）提高查询的多级索引能力。基于数据模型索引主键（key）和时间范围的查询虽然能被高效地执行，但要提高其他属性的查询效率，还需进一步构建二级索引。

2.1 数据模型与假设

本文中数据元素以四元组$<x,y,t,e>$形式存在，其中x、y、t和e分别为经度、纬度、时间戳和有效负载。在数据插入之前，可在x和y上应用Z-order将输入数据元组$<x,y,t,e>$转换为$<z,t,e>$，其中z为位置键（以下简称为键）。本文假设元组以其时间戳的递增顺序到达，用户查询基于一系列键和时间戳的建立，使得时间和键形成二维空间R。将一段范围的时间和键分别称为时间间隔和键间隔，若给定任意时间间隔和键间隔，则在R中可唯一确定一个矩形区域。

2.2 模型结构

本文提出的Tars模型可在一组分布式服务器集群上运行并通过局域网络互连，其结构如图1所示。其中，消息中间件传递的实时流数据是数据来源，其由数据入口处理层（数据调度层）接收，再根据数据分区划分规则分发到下游索引服务层。索引服务层将实时流数据插入到模板B+树中作为索引结构，在其超过阈值后进行分组压缩，并以数据块形式写入分布式文件系统CephFS中。元数据管理服务器通过zookeeper和R树[21]维护系统的状态，其中包括数据调度层对数据的分区模式和数据块的元数据信息。

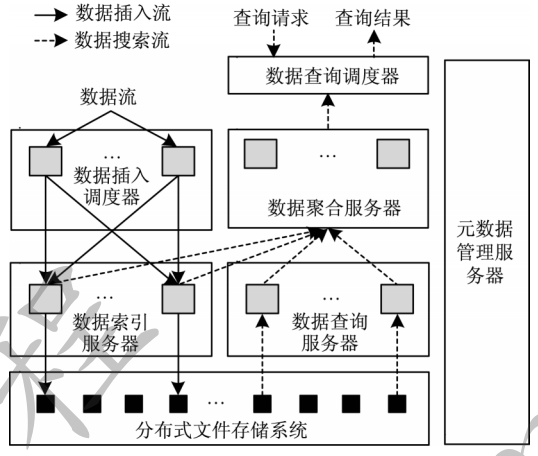

图1 Tars模型结构

Fig.1 Tars model structure

本文模型支持多用户查询的并发处理，查询调度层根据查询标准和元数据服务层的信息将用户查询转换为独立子查询，并在索引服务层和查询服务层间并行执行，然后将查询结果返回到各自聚合服务器进行聚合函数处理，再合并返回给对应用户。下文分别介绍高吞吐量数据插入中使用的数据索引存储方法（见图1中实线指示的路线）和实时查询处理方法（见图1中虚线指示的路线）。

3 流式数据存储与查询模型的实现

3.1 数据索引和压缩存储原理

常用的内存索引技术包括B+树、LSM树以及bulk loading等。本文场景中需要插入大量的实时数据元组，而B+树在节点插入时要进行分裂，在插入海量数据元组时会带来较大分裂开销，导致效率降低。LSM树需不断将两层索引进行合并，存在较大时间开销，不适用于实时应用场景。bulk loading技术通过批量插入数据元组可减少节点分裂导致的性能下降，但其更适用于批处理场景，而不适用于实时数据查询。因此，本文对模板B+树进行改进，以提升索引的压缩存储与持久化能力。

在数据写入方面，本文数据以时间事件为驱动，并要求数据具有实时可见性，因此采用范围分区（根据键值范围和时间范围进行划分）的方式将不同范围的数据以模板B+树形式写入内存中并构建索引，以增强在内存中快速写入与查询能力。当数据在B+树索引中达到预设值大小（如16 MB）时，考虑其持续增长会影响内存索引效率并增强数据持久化能力，因此需要将数据以数据块形式压缩后写入底层分布式文件系统中。当查询范围和数据块文件的区间范围有交集时，系统将从文件系统中读取并解析

计算机工程 www.ecice.com

数据块，并检索出符合查询条件的数据元组。

数据块的设计布局是一个重要环节，合理的布局能有效减少查询时解析的数据量。例如，当一个查询只覆盖部分数据块文件时，若采用较合理的数据块布局，则无需从文件系统中读取整个数据块就可访问到所需要的数据元组。图2为数据块文件的存储结构，其中包括template索引层和compressed chunk压缩数据层。索引层存储模板B+树的非叶节点部分会按照树的层级自上到下且自左到右的连续存储。每个节点均额外记录了key数组对应孩子节点的偏移量。当该子节点为非叶节点时，偏移量为指向索引层的一个数组；当该子节点为叶节点时，偏移量指向压缩数据层并分为两个数组，即该子节点所在叶节点分组的组内和组间偏移量。

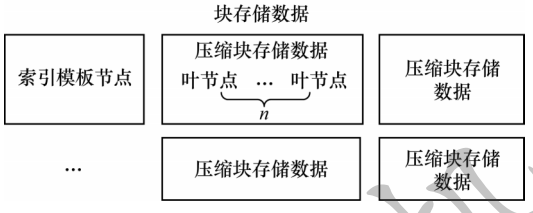

图2　数据块文件存储结构

Fig.2　Storage structure of data block file

数据层采用分组并压缩的布局形式从左到右有序存储模板B+树全部叶子节点，根据预设的组容量k，从最左侧叶节点开始以k个为一组进行压缩，生成压缩数据块，一直压缩到最右侧叶节点。每个压缩数据块互相独立，并记录组内叶节点中数据元组的键值范围，从而使压缩数据块在满足查询条件覆盖范围的同时，能根据偏移量进行针对性读取，以避免读取整个数据块。

压缩数据块中叶节点布局包括索引部分和数据部分，如图3所示。索引部分为一个key数组，包含与数据元组对应的一个数组偏移量。数据部分为一个数据元组数组，其存储了流入系统的原始数据元素。在获取叶节点后，使用二分法在索引部分key数组中找到数据元组对应的偏移量，然后根据偏移量找到数据部分对应的数据元组，即为查询结果。

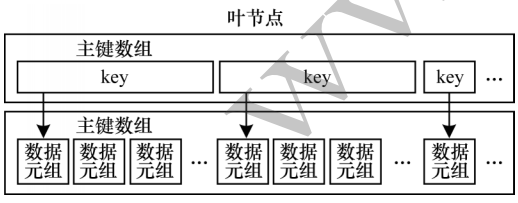

图3　叶节点布局结构

Fig.3　Leaf node layout structure

因为数据块文件所存储数据（模板B+树）的键值分布会随插入元组的变化而改变，所以元组并不总是稳定地平均分布在叶节点中。如果模板B+树在保持整体分布稳定的同时，部分叶节点的溢出元组较其他叶节点仍较多，则会给数据块文件的压缩解压带来额外开销。由于不同叶节点分组采用并行压缩，如果某个分组的叶节点数据过多未被处理完毕，则将导致整个数据块持久化过程阻塞和时间开销增多，从而造成计算资源的浪费，因此为使模板B+树在应对不稳定叶节点分布而进行压缩存储时更有鲁棒性，本文提出一种模板B+树在持久化时的分组压缩方法，用来计算叶节点分组时每组应分配的叶节点个数。

假设系统所在服务器的线程数预设值为m（即系统可并发处理的叶节点压缩组个数），$L(\ell^-,\ell^+)$为模板B+树的叶节点区间，则模板B+树的叶节点范围$P=\{\ell_1,\ell_2,\cdots,\ell_N\}$，$N$为模板B+树的全部叶节点，$D$为模板B+树的全部数据元组。在$L(\ell^-,\ell^+)$和$U_{1\leqslant i\leqslant N}L_i$中，对于任意的$i\neq j$，$L_i$与$L_j$交集为空。因此，每个叶节点分组中分配较理想的叶节点个数为：

$$K=|N|/m \tag{1}$$

当分组中实际存储的数据元组数量大于或等于树中的一系列元组之和的比率$J=2/|D|$时，可认为当前叶节点分组不适用。为适应当前数据元组分布的范围，需重新规划叶节点压缩的分组。

利用叶节点分组个数m和当前模板B+树的全部数据元组个数D，结合数据元组在模板B+树中的分布范围，可重新确定叶节点的分组结构。对于模板B+树而言，叶节点键值从左向右依次递增，如果用V表示数据元组的键值数组，用$V[i]$表示该数组中的第i个元素，则可直观地为元组键值平均分配K个新划分范围$Q=\{V_1,V_2,\cdots,V_k\}$，V_1的范围表示为：

$$V_i=\begin{cases}[v^-,v(in+1)], & i=1\\ [v(i-1)n+1],v(in+1)), & 2\leqslant i\leqslant l-1\\ v[(i-1)n+1,v^+], & i=l\end{cases} \tag{2}$$

根据新元组分组划分范围Q和式（1）的叶节点理想分组范围K，可重新调整当前叶节点分组并构建新的压缩数据文件。假设系统线程数m=3，当前有一棵具有6个叶节点的模板B+树，这棵B+树存储[0,20)范围内的数据，数据范围划分为：$V=$ {[0,4),[4,7),[7,10),[10,13),[13,16),[16,20)}，且当前模板B+树的叶节点实际上已插入数据元组集合$P=$ {[2],[4],[7,8,9],[10,11,12],[17]}，树的大小已达到阈值，需要存储到底层的分布式文件系统。根据式（1）计算得到叶节点分组结果$K=3$，则叶节点被分为3组，在每组中，$Q_1=\{2,4\}$，$Q_2=\{7,8,9,10,11,12\}$，$Q_3=\{17\}$，其中Q_2的数据元组占总数据元组的比率$J=2/3$，与当前模板B+树的J相等，因此根据式（2），对数据元组重新划分范围得到：$Q_1=\{2,4,7\}$，$Q_2=\{8,9,10\}$，$Q_3=\{11,12,17\}$。对应到叶节点中，即：将数据元组集合P的第3个叶节点的元组{7}拆分到第1个叶节点分组Q_1中，将P的第4个叶节点的元组{11,12}拆分

www.ecice06.com

到第3个叶节点分组Q_3'中，并在压缩时记录其所在分组的数据块文件字节偏移量与组内偏移量，从而在查询时进行访问。

当一个查询分解为独立的子查询后，为查找已存储到分布式系统中的数据元组是否符合查询条件，系统需在数据块索引层中获取满足查询条件的元组偏移量。根据数据元组所在叶节点的组间偏移量，先找到其对应的叶节点分组并解压，再根据数据元组在叶节点中的组内偏移量找到数据元组。为提高数据访问的局部性，数据元组的key在索引层的存储顺序与数据元组在数据层的存储顺序一致。

3.2 二级索引支持

为获得查询所覆盖的数据区域集合，元数据服务器使用R树来存储数据区域。当系统接收到查询$q=\langle k_q,t_q,f_q\rangle$（$k_q$和$t_q$分别为查询的key区间和时间区间，$f_q$为谓词函数）时，查询服务器会访问元数据服务器中的R树，并获取一组查询q所涵盖的区域R_q。每个数据区域$r_i=\langle k_i,t_i\rangle\in\mathbb{R}_q$，其中$k_i$和$t_i$分别为数据区域的key区间和时间区间，生成子查询$q_i=\langle k_i\cap k_q,t_i\cap t_q,f_q\rangle$并发送给相应的索引服务器或查询服务器进行处理。

根据数据区域划分模式，模型中基于key和时间范围的查询能被高效地执行（见2.1节），即可用key和时间属性为主索引来组织数据的分布。当查询条件涉及到key和时间外的其他属性时，为避免对结果数据进行遍历并提供在非主键查询上的高效索引和快速查询能力，需建立对应属性的二级索引，并将索引表保存在本地缓存与键值对数据库中，以获得更好的可扩展性和容错能力。在实际应用中存在查询多个非主键属性组合的要求，因此，类似于数据库中的多字段索引，对于多个非主键属性列的组合查询情况，本文基于多个非主键属性列建立组合索引。

在轨迹流数据的应用场景中，用户会根据经度和纬度等基本信息构建一个查询条件，其中经度和纬度区间k_i、时间范围区间q_i等键值属性范围均可划分，因此查询步骤如下：1）对查询进行分解，构建独立区间的子查询q_i（其具有相同的谓词函数f，可为用户提供非主键组合查询条件）；2）系统对数据进行分层索引，数据在未达到内存阈值前，先存储于内存的多个模板B+树中，当超过内存阈值后，再压缩存储于持久化文件系统CephFS内，由于两者均可在不同服务进程中进行并发处理，因此如何充分利用并发查询的能力也是本文考虑的问题。考虑到非主键组合索引的建立，在独立子查询分配到各服务进程之前，先通过非主键组合索引找到本次查询请求可能覆盖的内存B+树和持久化数据文件，再基于这部分数据进行模板B+树内部的数据索引查询，以减少不相关数据的查询开销。

在本文Tars模型中，考虑到复杂查询针对非主属性的查询条件，因此采用“是/否”的形式转换为“0”和“1”的问题。“1”表示有符合条件的数据，子查询可以进行下一步处理；“0”表示没有符合条件的数据，子查询可以被忽略。类似于文献[22-23]中Bloom过滤器与网络流量的结合应用，本文将预设的一个或多个数据元组属性通过Bloom过滤器的k个Hash函数映射到位数组中k个位置上，并将这k个位置的值均设置为1，表示该属性值可能存在于数据块文件中。

设ε为Bloom过滤器的最大误判率，N为集合元素个数，m为Bloom过滤器位数组长度，k为Bloom过滤器Hash函数的最优个数，q为查询时分解的一系列独立子查询，data为流入系统的数据元组，array为组合索引对应的位数组，QList为最终需要执行的子查询列表，则对本文模型二级索引算法描述如下：

算法1 Tars模型二级索引算法

输入 最大误判率ε，集合元素个数N，子查询q

输出 是否将子查询q放入查询列表Result

```
1.m,k←Compute Hash(ε,N)
2.array←In it Hash Array(m,k)
3.array←Indexing(data)
4.IF Has Persistence(q)
5.Return Result = False
6.ELSE
7.IF Array Has Q(q)
8.QList←q
9.Return Result = True
10.ELSE
11.Return Result = False
```

上述算法的具体步骤如下：

1）采用Compute Hash操作通过预设最大误判率ε、期望集合元素个数N计算出位数组长度m和Hash函数最优个数k。

2）通过In it Hash Array操作根据参数k和m构建Bloom过滤器，并将一维数组的值均初始化为0。

3）当流入系统的数据元组插入到模板B+树时，利用Indexing操作将其二级索引属性值映射到对应的Bloom过滤器中。

4）对于查询分解的独立子查询条件，使用Has Persistence操作判断该查询范围的数据是否已写入文件系统中，如果是，则执行步骤5，返回查询结果为真；否则执行步骤6。

5）当所查询数据仍缓存于内存中时，无需对数据块文件进行过滤。

6）当查询范围数据所在的内存索引已达到阈值并写入文件系统中时，如果指定属性的值存在于对应的Bloom过滤器中，则利用Array HasQ方法，将子查询q放入查询列表QList准备进行下一步查询处理，同时返回查询结果为真；否则对子查询进行过滤，并返回查询结果为假。

计算机工程 www.ecice06.com

4　实验与结果分析

本文Tars模型采用流数据处理系统的拓扑结构并基于Apache Storm分布式流数据处理系统来实现。在该模型中，各服务层分别是拓扑结构中的不同组件，这些组件通过自定义路由规则进行连接。其中，Storm负责组件的资源分配与数据传输通信，CephFS负责数据块文件的分布式存储。

本文实验使用真实数据集T-drive[24]。实验在16台t2.2xlarge亚马逊EC2集群中进行，每台机器运行2个数据调度服务、2个索引服务、1个查询调度服务和2个查询服务层服务。通过采取集群统一配置，可避免实验过程中内存、CPU、网络等机器配置对实验结果的影响，实验相关配置信息如表1所示。

表1　实验配置信息

Table 1　Experiment configuration information

配置名称	配置型号
操作系统	UbuntuLinxu16.04LTS
硬件配置	8核3.4 GHz，16 GB内存
网络带宽/(GB·s^{-1})	1
Storm版本	1.1.0
CephFS版本	0.80.6

4.1　索引压缩存储的性能评估

在块存储数据文件(StoreFile)进行数据层压缩存储时，将不同条件下各种压缩方法的索引压缩性能进行比较，结果如图4所示。图4(a)为不同压缩方法下模型的数据插入速率变化情况。可以看出，与StoreFile不压缩直接写入到文件系统相比，压缩后存储的数据插入速率明显提高，这是因为数据压缩后减少从内存到文件系统的磁盘I/O读写时间，且小数据容量的压缩时间较少，缩短了数据写入时导致该服务器数据插入工作的停滞时间。

由图4(b)和图4(c)可以看出，在不同的key选择率和不同查询时间范围(距离请求最近5 s、60 s和300 s)内，查询时延在进行数据压缩处理后明显降低。根据查询键值范围，在元数据层找到StoreFile压缩时数据层每个分组偏移量和组内每个叶节点偏移量，就可跳过无效检索数据，仅读取指定叶节点数据，从而降低时延。此外，Snappy压缩方法在数据插入和查询时有较好的性能表现，其原因是该压缩方法适合大量数据传输场景，数据压缩速度是其他压缩方法的1.5倍～1.7倍，而本文模型涉及到数据的内存索引，需压缩存储到文件系统并进行实时数据访问，要求数据传输通信效率较高，且Snappy压缩方法不会占用大量CPU，当本文模型混合负载复杂流数据进行统计聚合工作时，在资源方面对任务影响较小，保证了查询时延的稳定性。

图4(d)为不同压缩方法压缩率的对比情况。可以看出，GZIP作为CPU密集型方法压缩率较高，但由于其会影响模型对数据的计算处理，因此在数据存储和查询性能上均表现较差。Snappy压缩率相对较低，但能满足本文模型对数据文件快速压缩解压以提高数据在拓扑结构中快速流转的场景要求，其作为StoreFile的数据分组压缩存储方法，具有较好的性能表现。

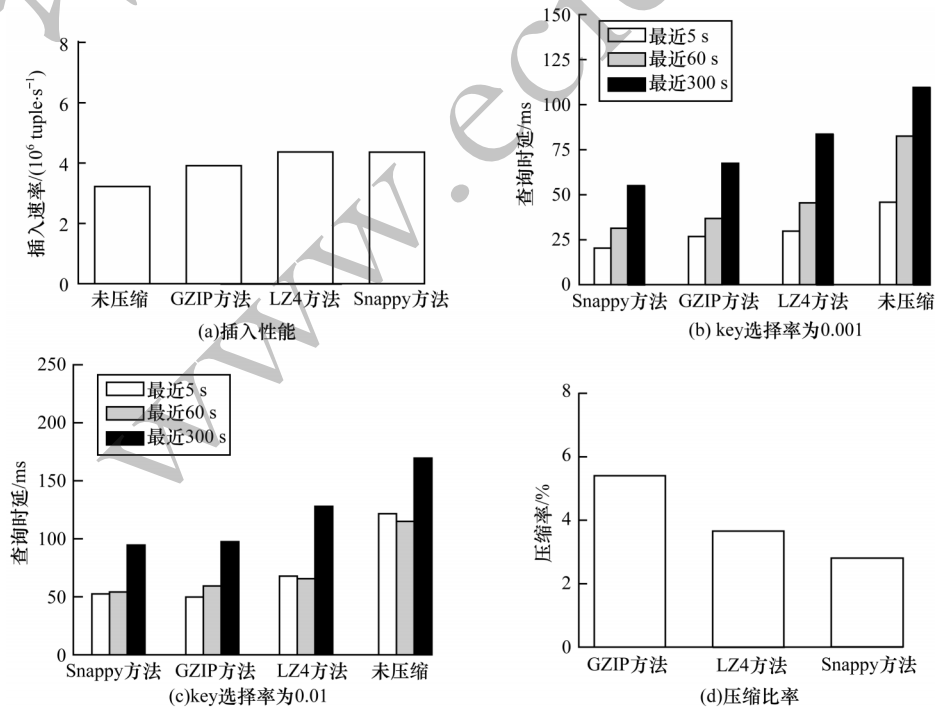

图4　不同条件下不同方法的索引压缩性能

Fig.4　Index compression performance of different methods under different conditions

计算机工程 www.ecice.com

图5为不同StoreFile存储容量和键范围对本文模型的数据压缩存储性能影响的评估结果。由图5(a)可以看出,当StoreFile容量小于32 MB时,模型数据写入速率随容量增大而提高,其原因是降低内存索引服务器中StoreFile达到阈值后落盘到文件系统的频率,缩短系统磁盘I/O读写时间与数据插入索引停止时间(写文件时该内存索引服务器的数据插入索引会暂停,直到数据完成在文件系统的写入)。当StoreFile容量超过32 MB后,模型数据写入速率随容量增大而降低,这是因为StoreFile所需压缩存储时间成本较高,导致数据插入索引工作停滞状态过长。图5(b)为本文模型在不同键范围与StoreFile容量下查询时延的变化情况。可以看出,查询时延随着StoreFile容量增大而升高,其原因是每个StoreFile均有不同的键值范围和时间域范围,而根据压缩存储形式可以仅读取StoreFile中指定的一系列叶节点以及叶节点中指定的部分数据,因此,对于给定键值范围的独立子查询,其数据读取范围与StoreFile容量成正比例增长。当StoreFile容量接近8 MB时,数据查询时延趋于稳定,这是因为当StoreFile容量较小时,数据压缩比较低,压缩所需时间和压缩后StoreFile的大小变化较小,而CephFS底层为OSD块存储形式,其读取块数据存在访问时延,当StoreFile容量较小时,该访问时延会占较大比例,使得查询时延趋于平稳。由上述分析可知,当StoreFile容量取值为8 MB时,在存储和查询上具有更优的数据压缩性能。

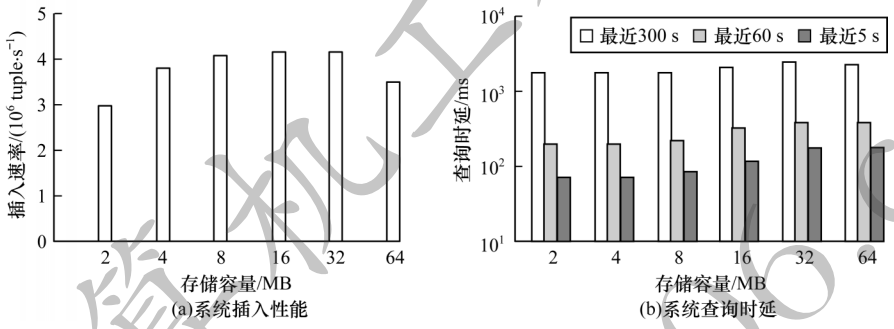

图5 本文模型压缩存储性能评估结果

Fig.5 Performance evaluation results of compressed storage of the proposed model

4.2 复杂查询条件下的查询性能评估

在查询性能评估引入二级索引方式后,在带有谓词函数等复杂条件下,将本文Tars模型和WaterWheel模型的查询性能进行比较,结果如图6所示。其中,WaterWheel获取所有符合键值范围和时间范围的查询结果并返回到查询调度层,然后统一通过谓词函数条件来串行过滤处理结果,其不支持多用户查询的并行调度处理。由图6可以看出,本文模型在不同时间范围下查询延迟较WaterWheel更小,且随着key选择率的增加,该性能差距更明显,这是由于WaterWheel不支持查询分解时对子查询的二级索引,需读取全部符合key和时间范围的内存索引与分布式文件数据块,再统一进行谓词函数条件过滤,并在查询调度层对多用户查询结果进行串行化处理后返回给用户,因此查询时延较高。本文模型使用Bloom过滤器建立谓词函数条件值与位图数组的映射关系,通过对每个独立子查询进行二级索引,能提前判断是否存在满足该索引的StoreFile,如果不存在,则直接将子查询进行过滤,减少了无效查询时间。

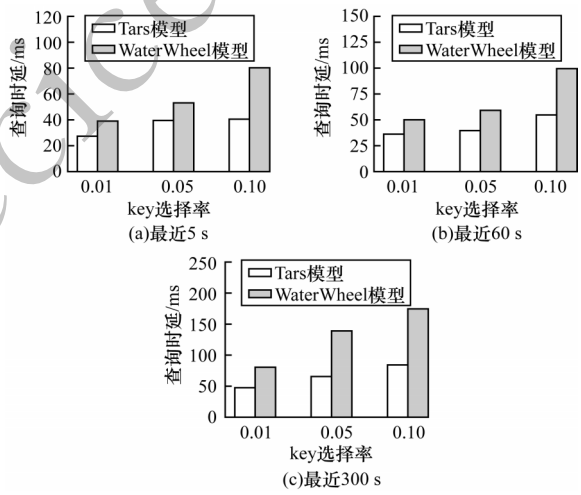

图6 不同时间范围内2种模型的二级索引查询性能对比

Fig.6 Performance comparison of two models for secondary index queries in different time ranges

图7为查询分解后通过二级索引确定(经Bloom过滤器过滤)的有效和无效子查询的百分比(简称为二级索引命中百分比)。通过本文模型的二级索引检测可知,在带有复杂谓词函数的查询中,满足键值范围和时间范围的子查询超过80%为无效子查询,本文忽略这些无效子查询。上述结果验证了模型支持二级索引的重要性。

计算机工程 www.ecice.com

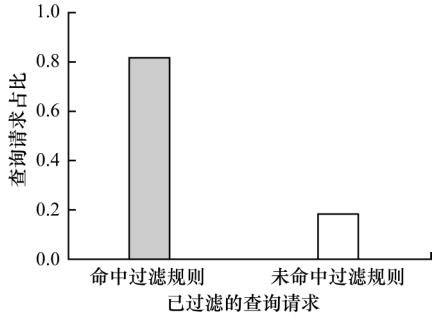

图7　本文模型二级索引命中百分比

Fig.7　Percentage of hits in the secondary index of the proposed model

在不同时间范围和key选择率下将本文Tars模型与HBase、WaterWheel模型在T-drive数据集上的查询性能进行对比，结果如图8所示。其中，HBase、WaterWheel模型在底层均使用HDFS（大数据解决方案通用的分布式文件系统，支持海量数据离线批处理）作为分布式文件系统。为保证3种方法的查询性能在相同条件下可进行比较，将插入速率统一设置为每秒50 000个元组（HBase最大插入速率的一半）。

由图8可以看出，Tars在不同的key选择率与时间范围下查询延迟均少于HBase和Waterwheel。随着key选择率不断提升，HBase与其他两种模型的查询延迟差距逐渐增大，其原因是HBase不支持在非key属性上的范围索引，其需读取全部符合key选择率的元组并测试其是否符合时间范围，造成查询延时较高。本文模型在全局划分出二维区域R，并将查询分解为独立子查询，经过二级索引处理后，可过滤掉不符合查询条件谓词函数f的StoreFile，从而减少查询时延。WaterWheel不支持二级索引，其使用HDFS作为底层文件系统，在处理实时数据任务时基本时延较高，而Tars采用增量式存储形式处理数据，其历史数据变更较少，无需进行目录结构维护，因此将CephFS作为文件系统能加大数据压缩存储容量并提升多级索引的效率。

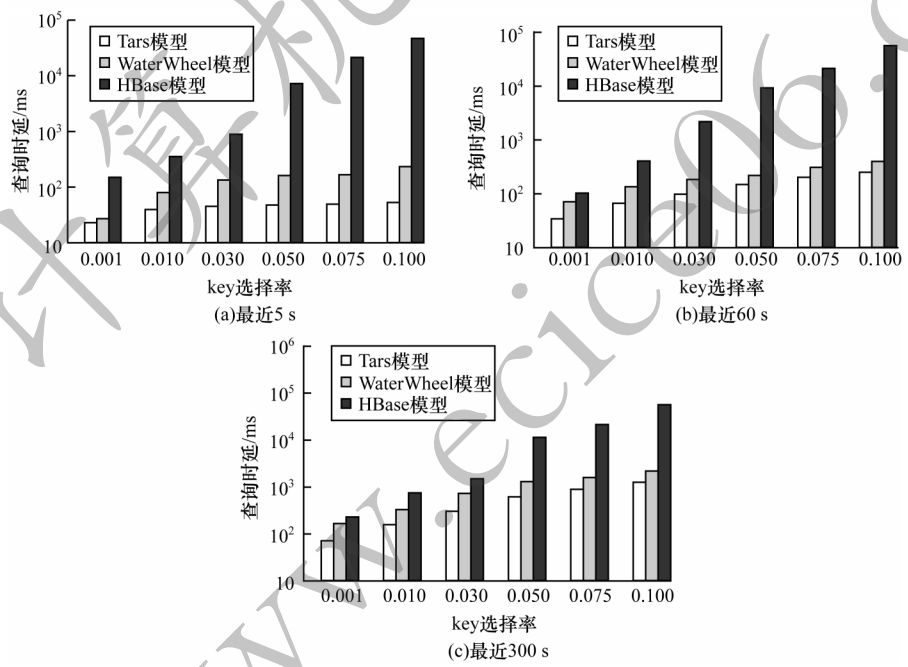

图8　不同时间范围和key选择率下3种模型的查询性能对比

Fig.8　Query performance comparison of three models under different time range and key selection rate

5　结束语

本文提出一种面向轨迹流数据的压缩存储和多级索引方法，构建数据分区和内存索引并分组压缩存储到分布式文件系统以提高模型存储效率，采用流数据多级索引方法，保证复杂条件函数下查询分解的稳定性。实验结果表明，与传统HBase、WaterWheel等方法相比，该方法具有更高的数据存储性能与查询效率。后续考虑将承载模型数据传输和网络通信的拓扑结构Apachestorm模型替换为微服务模型，解决网络数据传输速率受带宽限制的问题。

参考文献

[1] TU Xuezhen, TU Yaofeng, CHEN Xiaoqiang. An optimized Key-Value NoSQL system [J]. Computer Engineering, 2019, 45(6): 52-59. (in Chinese)
屠雪真，屠要峰，陈小强．一种优化的Key-Value型NoSQL系统[J]．计算机工程，2019，45(6)：52-59.

www.ecice06.com

[2] GEORGE L. HBase: the definitive guide random access to your planet-size data[M]. Sebastopol, Russia: O'Reilly Media, 2011.
[3] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: amazon's highly available key-value store[J]. ACM SIGOPS Operating Systems Review, 2007, 41(6): 205-220.
[4] 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): 4-6.
[5] WANG Li, ZHOU Minqi, ZHANG Zhenjie, et al. Elastic pipelining in an in-memory database cluster[C]//Proceedings of 2016 International Conference on Management of Data. New York, USA: ACM Press, 2016: 1279-1294.
[6] YANG F, TSCHETTER E, LEAUTE X, et al. Druid: a real-time analytical data store[C]//Proceedings of 2014 International Conference on Management of Data. New York, USA: ACM Press, 2014: 157-168.
[7] ANDERSEN M P, CULLER D E. BTRDB: optimizing storage system design for timeseries processing[C]// Proceedings of the 14th Conference on File and Storage Technologies. Santa Clara, USA: USENIX Association, 2016: 39-52.
[8] AGUILERA M K, GOLAB W, SHAH M A. A practical scalable distributed B-tree[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 598-609.
[9] ONEIL P, CHENG E, GAWLICK D, et al. The log-structured merge-tree[J]. Acta Informatica, 1996, 33(4): 351-385.
[10] SEARS R, RAMAKRISHNAN R. BLSM: a general purpose log structured merge-tree[C]//Proceedings of 2012 International Conference on Management of Data. New York, USA: ACM Press, 2012: 217-228.
[11] TAN W, TATA S, TANG Y, et al. Diff-index: differentiated index in distributed log-structured data stores[C]// Proceedings of the 17th International Conference on Extending Database Technology. Berlin, Germany: Springer, 2014: 700-711.
[12] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system[J]. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35-40.
[13] GOLAN G G, BORTNIKOV E, HILLEL E, et al. Scaling concurrent log-structured data stores[C]//Proceedings of the 10th European Conference on Computer Systems. New York, USA: ACM Press, 2015: 1-14.
[14] KESARWANI M, KAUL A, SINGH G, et al. Collusion-resistant processing of SQL range predicates[J]. Data Science and Engineering, 2018, 3(4): 323-340.
[15] MA L, VAN A D, HEFNY A, et al. Query-based workload forecasting for self-driving database management systems[C]// Proceedings of 2018 International Conference on Management of Data. New York, USA: ACM Press, 2018: 631-645.
[16] ISLAM N S, LU X, WASIRURAHMAN M, et al. Triple-H: a hybrid approach to accelerate HDFS on HPC clusters with heterogeneous storage architecture[C]// Proceedings of 2015 IEEE International Symposium on Cluster, Cloud and Grid Computing. Washington D. C., USA: IEEE Press, 2015: 101-110.
[17] KRASKA T, BEUTEL A, CHI E H, et al. The case for learned index structures[C]//Proceedings of 2018 International Conference on Management of Data. New York, USA: ACM Press, 2018: 489-504.
[18] WANG Li, CAI Ruichu, HE Jiong, et al. Waterwheel: realtime indexing and temporal range query processing over massive data streams[C]//Proceedings of the 34th IEEE International Conference on Data Engineering. Washington D. C., USA: IEEE Press, 2018: 21-27.
[19] MAZUMDAR P, WANG L, WINSLET M, et al. An index scheme for fast data stream to distributed append-only store[C]//Proceedings of the 19th International Workshop on Web and Databases. New York, USA: ACM Press, 2016: 31-36.
[20] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: a scalable, high-performance distributed file system[C]// Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Santa Clara, USA: USENIX Association, 2006: 307-320.
[21] BECKMANN N, KRIEGEL H P, SCHNEIDER R, et al. The R*-tree: an efficient and robust access method for points and rectangles[C]//Proceedings of 1990 International Conference on Management of Data. New York, USA: ACM Press, 1990: 322-331.
[22] MITZENMACHER M. Compressed bloom filters[J]. IEEE Transactions on Networking, 2002, 10(5): 604-612.
[23] ZHAI Jinfeng, SUN Libo, LU Kai, et al. Research on flow sampling algorithm based on Counting Bloom Filter[J]. Computer Engineering, 2018, 44(8): 273-278. (in Chinese)
翟金凤, 孙立博, 鲁凯, 等. 基于Counting Bloom Filter的流抽样算法研究[J]. 计算机工程, 2018, 44(8): 273-278.
[24] YUAN Jing, ZHENG Yu, XIE Xing, et al. Driving with knowledge from the physical world[C]//Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining. New York, USA: ACM Press, 2011: 316-324.

编辑 宋 圆

计算机工程 www.ecice06.com