一种基于 Spark 的分布式时态索引方法

郑晓东 王 梅 陈德华 张碧莹
(东华大学计算机科学与技术学院 上海 201620)

摘 要 基于 Spark 分布式计算平台提出一种分布式时态索引方法。该方法提出时态数据集的分段索引构造策略,对每一分段设计基于 Spark 的时态索引构建方法及基于 Spark RDD 的并行查询策略;根据时态查询所涉及的 Spark RDD 分区模式的不同,将其分为分区独立查询,跨区查询以及跨段查询,并分别针对不同模式的时态查询提出优化的辅助索引结构,提高查询效率;在基准数据上进行实验,验证了所提索引策略的实用性和高效性,同时表明所提方法对数据规模的有效自扩展性以及降低了集群硬件配置需求。

关键词 时态数据 时态索引 Spark 分布式 分段存储

中图分类号 TP391 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2018.05.018

A SPARK-BASED DISTRIBUTED TEMPORAL INDEXING METHOD

Zheng Xiaodong Wang Mei Chen Dehua Zhang Biying
(*College of Computer Science and Technology, Donghua University, Shanghai 201620, China*)

Abstract Based on the Spark distributed computing platform, a distributed temporal indexing method is proposed. Firstly, this paper proposes a segment-indexing strategy of temporal data set, and designs Spark-based tense index construction method and Spark RDD-based parallel query strategy for each segment. According to the Spark RDD partition pattern involved in the temporal query, it is divided into domain independent query, cross-domain query, and cross-segment query, and proposes the optimized auxiliary index efficiency. Finally, the experiments on benchmark data verifies the practicability and efficiency of the proposed indexing strategy, indicating the method is effective on the scalability of the data scale and reduces the cluster configuration requirements.

Keywords Temporal data Temporal index Spark Distributed Segmented storage

0 引 言

大数据时代已然来临。时间作为客观事物的固有属性,几乎所有信息都显式或隐式地具备时态特征。大数据的产生往往也是经过时间累积形成,因此大数据也被理解为针对某个对象在时空两个维度上的“全息”数据。若对数据进行时态的关联分析,有望挖掘出数据的巨大价值,从而使得研究和使用数据的“时态”信息意义日显突出,对时态数据管理查询技术的研究十分迫切。

在考虑数据的时态信息时,被打上时间标签的数据,会伴随着时间的延伸而生效或失效。因此,对于一个实体来说,它会拥有多个数据记录,很明显传统默认时间点的静态“快照”式数据存储模式难以满足时态查询需求。一些时态数据库模型被相继提出,如 TimeDB[1] 等。与此同时,Oracle、IBM DB2[2]、SAP HANA[3] 等知名的商业数据库也开始支持一些简单的时态查询功能,包括时间旅行,时态聚合以及时态连接等。以时态连接为例,检索“在相同时间点中,两个银行账户数据表中存款相同的客户存款额度”,需要在时间以及对象两个维度上进行双向筛选,如何在庞大的数据集进行快速、高效的时态操作执行成为一个十分有挑战性的问题。

索引是数据库中加速查询的有效技术。为此,研究者从不同角度为时态数据模型提出了多种时态索引

收稿日期:2017-10-09。上海市科技创新行动计划(16JC1400802)。郑晓东,硕士生,主研领域:数据库技术。王梅,教授。陈德华,副教授。张碧莹,硕士生。

技术。文献 [4] 提出的基于 B + 树 Time Index 索引以及文献 [5] 中提出的 Historical R 树索引对传统树型索引进行扩展已支持时态数据,文献 [6] 针对时态 XML 数据,构建不同的时态索引模型以支持时态查询。近来,SAP HANA 提出的 Timeline 索引 [7] 以及双时态索引技术 [8] 采用顺序结构实现高效的时态查询。然而,目前所设计的大部分索引均在单机环境下实现,当数据集规模较大时,上述方法将花费较高的代价在遍历庞大的索引树或索引队列上。同时,其索引结构和查询算法的执行效率对机器性能的要求极高,以支持 SAP HANA 的 Timeline 时态索引为例,其执行默认的机器内存达 192 GB [7]。上述问题极大地限制了索引技术的普遍应用和查询效率的提升。

在数据规模不断增长的今天,越来越多的应用都是基于存储在分布式系统中的大规模数据。Spark 分布式处理框架通过将海量数据分发至各个计算节点,并基于弹性数据集 RDD 及 MapReduce 技术极大地加快了数据的处理速度 [9]。然而该框架中并没有对 hdfs 中文件的行级记录进行索引,本文基于 Spark 分布式计算平台设计并实现了一种分布式时态索引方法。该方法首先提出时态数据集的分段索引构造策略,进一步设计基于 Spark 的时态索引构建方法及基于 Spark RDD 的并行查询策略。本文主要内容如下:

1) 以 Timeline 索引为例,提出时态索引在 Spark 集群上的构建方法,设计分布式时态索引的并行查询框架。根据时态查询所涉及的 Spark RDD 分区模式的不同,将其分为分区独立查询,跨区查询以及跨段查询,并对不同模式的时态查询提出了优化的分布式查询算法,提高查询效率。

2) 优化辅助索引结构,加快"给定时间点查询"这一原子操作的效率,从而极大地提升时态聚合,时态旅行等分区独立查询的查询效率。进一步针对跨 RDD 分区及跨段的时态连接复杂查询,分别设计复合索引结构辅助查询,确保本文方法对数据规模的有效自扩展性同时降低集群硬件配置需求。

3) 在基准数据上,进行本文所提方法的有效性验证,证明了本文提出的索引策略的实用性和高效性。

1　相关工作

1.1　时态索引

传统数据库中索引技术是提高查询效率的关键技术之一,同样对于时态数据,合适的索引结构将有效地支持时态操作。时态数据索引大致可归为两类:树索引和队列索引。最早提出的时态数据索引结构 Time Index 就是基于 B + 树构建的,该索引针对每个时间间隔的结束时刻作为索引值建立 B + 树,每个叶子结点记录该时间点各事件的生效失效状态 [4]。但是该索引在时间跨度较大的情况下会引起较大的空间开销,同时这种基于 B 树的时态索引在构建时依赖于数据的时间有序性。另一种树形结构 R 树虽然最初的设计思想是为了索引空间数据,但是它被用来存储时间同样适合。文献 [5] 提出了 Historical R 树,每个时间戳都建立一个 R 树,且一个 Historical R 树中未发生变化的记录对应的结点只保存一个以节省空间。同时,除了这种以单一一个数据结构来索引全部数据外,还有以树形结构为基础,为每一个时间版本构建一个树来索引时态数据的方法,如 MVBT 索引 [10] 和 MV3D-RT 索引 [11]。相对于树形时态索引,基于队列的时态索引研究较少,文献 [7] 提出了 Timeline 时间线索引,该索引应用于 HANA,在内存中为时态数据建立一张时态表,每一个时态表对应一个 Timeline 时间线索引,在时间线上保存着各个数据的生效失效时间,该索引支持多种时态操作。

目前对于时态数据的操作主要分为时间旅行,时态聚合和时态连接。时间旅行是查询数据在过去或未来某个时间点或时间段所具有的状态,可以通过时间旅行回溯到某个时间版本的数据库状态。时态聚合和时态连接都是基于时间旅行所进行的操作,时态聚合是指对某个时间点或时间区间的有效记录进行聚合操作。时态连接是指对指定时间点下的有效记录进行连接操作。如今对于时态索引的研究还处于一个发展阶段,现阶段已提出的时态索引大多数仅仅支持有限的时态操作,部分更是仅支持时间旅行,无法有效地支持时态聚合和时态连接等复杂查询操作。

1.2　分布式索引

分布式计算框架如 Spark、Hadoop,采用的是分布式的文件系统,通过将大文件进行分片,切分成多个分片文件存储在多个存储节点上。平台在需要对该文件数据进行计算时,各个分片文件加载至多个计算节点上进行独立计算,计算子结果将返回主节点进行后续操作。这种存储计算模式虽然解决了大数据计算处理的效率问题,但是在对数据文件进行行级访问时,尽管数据文件的切分且分布式的读取加快了读取速度,其大量的时间浪费在了文件的寻道遍历上。Haojun Liao 提出了应用在 Hadoop 的分布式文件系统上的多维索引,考虑到 HDFS 中块大小、索引节点大小、网络中的数据通信等影响查询速度的因素,通过构建类似于 R

树的层次结构索引，避免查询时无意义的穷举减少查询响应时间[12]。Gankidi 等[13]提出通过构建 B + 树管理 HDFS 数据，索引维持在 SqlSever 中，用户可以通过 SQL 查询 HDFS 数据。Xie 等[14]提出建立在 Spark 平台上用于管理空间数据的两级索引结构，该索引维持在 RDD 中，区域索引保存空间区域中的数据信息，而全局索引管理着每个区域索引的边界信息。

总的来说，现有的时态索引技术尚处于研究阶段，大多数索引结构无法有效地支持时态操作，并且在面对大规模时态数据时，索引空间代价较大，效率变低，而传统的分布式索引大多是面向隐性时间属性数据，无法有效地支持对历史信息的相关操作。

2 基于 Spark 的时态索引

2.1 定义和符号

定义 1 时态表(Temporal Table)。包含具有时态属性的数据的表。

定义 2 时间区间(Period)。表示一个数据合法持续的时间，包括它的开始和结束。本文使用包含-排除方法来建模时间区间，使用 [start, end) 来表示。在实现过程中分别通过时态表的两列来分别表示记录的开始时间和结束时间。

定义 3 时间版本(versionID)。指时间区间内离散的时间点，这些时间点是单调递增的，本文将这些时间点称之为 versionID。

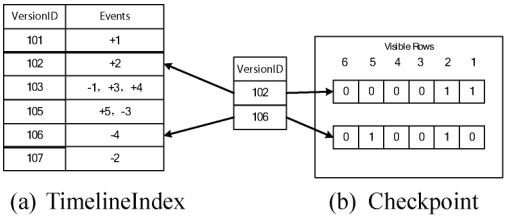
表 1 给出了一个时态表的例子。该实例模拟了一个银行系统，Alice 在 101 时间点在银行存入$ 100，Alice 在银行的存款为$ 100 这个状态在数据库中维持到了 103 时间点，即 103 时间点后(包含 103 时间点)第一条记录失效，第四条记录生效，表明 Alice 在 103 时间点在银行存入$ 500 使银行的存款到达$ 600。同理，Ann 在 102 时间点在银行存入$ 500，第二条记录在 102 时间点到 107 时间点有效，第三条记录表示 Grace 在 103 时间点在银行存入$ 300，直到目前为止，Grace 在银行的存款都为$ 300。

表 1 时态表

ROW_ID	Name	Balance	Start	End
1	Alice	$ 100	101	103
2	Ann	$ 500	102	107
3	Grace	$ 300	103	105
4	Alice	$ 600	103	106
5	Bob	$ 200	105	∞

2.2 索引结构

不论是基于树结构或是顺序结构的时态索引，均需在其索引结点中记录各时间点下事件的生效失效状态。以图 1(a)给出的 Timeline 索引结构为例，其中“+”表示该数据在对应时间版本下开始生效，“-”表示数据失效。上述索引可以有效地支持多种时态操作，然而其也存在着明显的缺陷。在执行时态操作时，由于索引只保存了各个记录的生效失效状态，如果要获取到某一时间版本下的有效数据，如查询“106 时间版本下的有效数据”，系统无法避免地需要从头遍历事务层至查询时间版本，以得到“106 时间版本下第二条记录及第五条记录有效”的信息。每一次的时态操作都会重复性的遍历索引，这种低效的重复性工作造成了极大的时间代价。



(a) TimelineIndex (b) Checkpoint

图 1 checkpoint 结构

因此，在 Timeline 时态索引中，同时维护检查点(checkpoint)辅助索引结构。checkpoint 中保存了当前时间下所有记录的有效状态，这种类似于快照的设计结构省去了在每次时态操作时需要从头遍历时间线索引所引起的大量开销，可以较为快速地获取指定时间版本下的有效记录，对于优化索引的检索效率至关重要。由于 checkpoint 的创建代价较大，一般在一个较大的时间间隔下创建一个新的 checkpoint 结构。

2.3 基于 Spark 的分段时态索引框架

Spark 上的所有数据操作都依赖于弹性分布式数据集(RDD)，RDD 被分布式的存储在集群的各个节点上，程序方能分布式的执行。给定当前时态表 T，由于时态表中数据是以时间特性排序的，为了保证各分区中数据的时态均衡性，应用 Spark 自带 Hash Partition 方法针对 ROW_ID 列进行哈希划分，从而得到 RDD 的各个分区。各 RDD 迭代本分区数据并行创建索引，因此 RDD 类型为一个二元组 (TT,TI)，其中 TT 即 Temporal Table 是本数据分区中的时态表，TI 即 Temporal Index 是针对于本分区时态数据构建的时态索引。很明显，随着时间推移，时态表 T 中的数据不断累积。为保证本文方法对数据规模的有效自扩展性。同时，为了避免当数据增大，RDD 在进行迭代计算时产生的内存溢出问题，首先将数据进行切分。给定当前时态表

$T = \{t_1, t_2, t_n\}$，n 为数据记录条数，将其切分为 $T = \{T_1, T_2, T_m\}$，m 为数据段个数。对每个数据段 T_i 按如上所述方法构建索引，通过设定 m 的大小以控制各段 RDD 的大小。同时，在 Spark 集群中维持全局索引来调度对应查询时段所涉及的各数据段索引 RDD 进行相关查询操作。

3　基于索引的时态查询执行

时态数据的查询操作可分为时间旅行、时态聚合、时态连接三种时态操作。根据操作所涉及的 Spark RDD 数据对象，本文将查询分为分区独立，跨区查询，跨段查询，如图 2 所示。并且对于每类查询，本文均针对性地提出优化的辅助索引结构及查询算法，以充分发挥分布式的优点，提高时态检索效率。

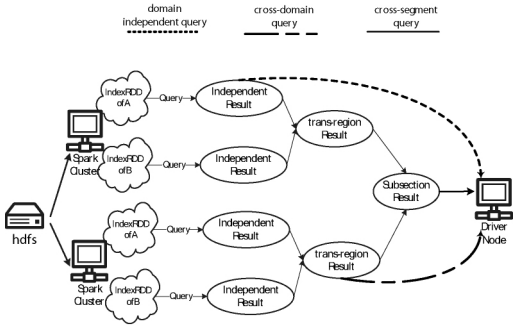


图 2　构建在 Spark 集群上的时态查询分类

3.1　分区独立查询

本文将时态索引构建在 Spark 集群中的 RDD 上，当集群读取至 RDD 上后，各个分区根据分区数据构建本分区索引。分区独立查询即指各个分区数据之间不必进行通信，各个分区在检索完本分区索引后直接返回结果集。

在几种时态操作中，时间旅行属于典型的分区独立型查询。时间旅行是查询某一时间版本 versionID 下数据库记录的状态信息，也就是得到该时间版本数据库的有效数据集。以时间旅行为例，详细描述分区独立查询在 Spark 集群的执行流程：在获取到待查 versionID 后，IndexRDD 各分区索引先检索本分区的 checkpoint 集合，获取到最邻近且早于查询时间版本的 checkpoint，根据 checkpoint 中的标志位查询对应行向量分段数据，获取到该时间版本下的有效数据。若 checkpoint 的时间版本和查询时间相同，该分区即可返回结果集；若与查询时间不等，则根据 checkpoint 的时间版本在时态索引中的位置信息接着向下遍历时态索引，直到遍历至查询时间在时态索引中的位置，整合计算得到该分区的有效数据集后返回给驱动节点。

传统的 checkpoint 结构需要遍历行向量来查看对应时间版本的数据库状态，由于 checkpoint 中会存储和时态表同等大小的行向量，若设时态表大小为 n，其遍历次数为 n。然而由于数据的时效性，在给定时间版本下同一对象的时态数据只有一条是生效的，且有效数据具有区域性（即一个时间版本下的有效数据大多会集中在行向量的一个分段内），造成大量的无效遍历。为此，本文首先利用 Spark 分布式和并行处理特点，对 checkpoint 结构进行优化，即对 checkpoint 中行向量进行固定长度的分段，并添加记录各分区有无有效记录的标记 flag 结构。如图 3 所示，checkpoint 分为标志位和有效数据位两部分，在有效数据部分按行号分段，并在标志位增设标志，记录该分段在指定时间版本下是否存在有效的行号。在具体查找时，即可根据标志位大量过滤无效记录。

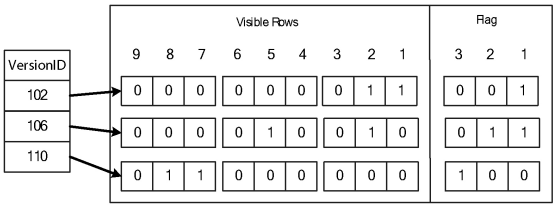


图 3　优化 checkpoint

优化 checkpoint 中的分段标志位简化了遍历复杂度，分段标志位为真再去遍历对应分段数据的有效性，若为假则直接跳过该分段。假定分段步长为 l，则时间复杂度近乎简化为 $(n/l + l)$，且各个分段可并行遍历，极大地降低获取 checkpoint 下有效数据的时间开销。同时，由于数据记录随时间的累计特性，在下一个时刻点对 checkpoint 创建时，仅需追加新增段，原有分段可重用，大大降低了 checkpoint 的创建复杂度。

3.2　跨区查询

跨区查询涉及到存储索引信息的 RDD 各分区数据间的通信，各分区间需要进行联合计算。时态连接就是最为典型的跨区查询，时态连接在时态操作中是一种较为复杂的操作，同时涉及到时间维度和对象维度的连接操作，如查询“在相同时间点中，两个银行账户数据表中存款相同的客户存款额度”。上述时态连接可以划分成两个阶段，第一个阶段利用时态索引以及 checkpoint 得到每个时间版本下两个数据集各分区的有效记录集合；第二个阶段是在相同查询点的有效记录集合中，根据查询条件以一定的连接规则对两个数据集的有效记录进行连接以得到查询结果。通过改进 checkpoint 结构虽然加快了第一阶段获取当前时间版本下有效记录的速度，但第二阶段低效的两两条件匹配付出了 $O(n^2)$ 的时间开销，尽管将索引应用在

Spark 集群，采用分布式的计算方法加快了匹配速度，但是存在集群中节点相互通信、数据拉取等问题，Spark 集群上的时态连接速度并不理想。

针对于时态连接过程中出现的低效的两两匹配以及大量 p 的数据通信问题，本文采用面向对象维度的辅助索引来优化时态连接效率。如图 4 所示，在通过第一阶段获取到有效记录后，在时态连接的第二阶段，各个数据节点不再进行数据广播，而是采用将全部有效数据进行混洗重分区，以各个记录的对象维度字段为 key 值进行 hash 混洗重分区，因此新的数据分区具有封闭性。即将连接的记录限制在本地分区，不用再进行数据通信来进行连接，这样的辅助索引结构不仅减少了数据通信引起的极大开销，而且缩减连接次数，提高了时态连接效率。具体的连接算法如算法 1 所示。

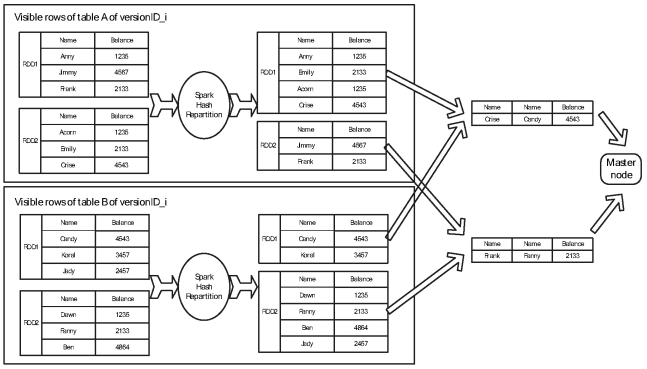


图 4 优化后的时态连接查询过程

算法 1 时态连接

输入：versionIDRange 时间版本范围，joinCondition 连接条件；

输出：Result 查询结果集；

```
1. for versionID_i in versionIDRange
2. RA,RA←temporalTravel(versionID_i);
                    /* 时间旅行获取数据表的有效记录* /
3. Repartition(RA) and Repartition(RB);
                    /* 以对象维度对 RA 和 RB 重分区* /
4. for partitionA in RA
5. for partition in RB
6. if partitionA. partitionNum == partition. partitionNum
7. Result←Join(partitionA,partitonB);
                    /* 拥有分区号相同的 RDD 进行连接* /
8. end if
9. end for
10. end for
11. end for
```

3.3 跨段查询

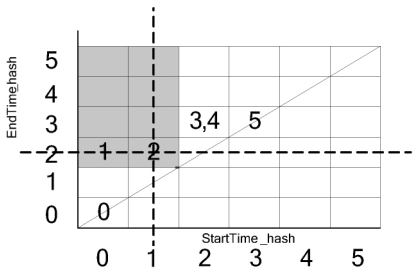
Spark 集群的一切操作都是基于 RDD 的。在根据数据集构建索引时，RDD 需要将所有数据都读进内存进行迭代计算，由于集群节点可以申请到的内存是有限的，当分区数据的计算量到达节点上限时就会内存溢出导致计算失败，为了避免这种情况，我们采用跨段的查询方式。

跨段查询就是对大数据进行分段处理，各个分段独立构建索引，将分段数据集及索引打包存储至 HDFS。考虑到时态数据具有时效性，每一个数据集同样有一个时效性，数据集中记录最早的有效时间以及最晚的失效时间即为该数据集的时间跨度。将时间跨度的相关信息设计到索引结构中将有效地加快查询速度，避免遍历过多的时间相错的数据集。因此，本文设计双 hash 结构来索引 HDFS 中的相关的索引文件，当执行时态操作时，由全局索引调配符合时间跨度的数据集及索引至 Spark 集群进行后续操作。

如图 5 所示，全局索引包括索引表和索引矩阵两部分，索引表保存各个索引结构的存储号、时间跨度，以及 HDFS 中的存储地址。索引矩阵将各个索引结构的存储号保存到对应时间跨度的块中，索引矩阵的横纵坐标为索引文件跨度的对应的 hash 值，横坐标为开始时间，纵坐标为失效时间。

Id	start	end	hdfs_address
0	0	987	hdfs://master:9000/indexFile/0/
1	578	2 578	hdfs://master:9000/indexFile/1/
2	1 224	2 345	hdfs://master:9000/indexFile/2/
3	2 437	3 689	hdfs://master:9000/indexFile/3/
4	2 577	3 964	hdfs://master:9000/indexFile/4/
5	3 257	3 996	hdfs://master:9000/indexFile/5/

(a) Index table



(b) Index matrix

图 5 全局索引

当进行时态操作时，先访问全局索引的索引矩阵，根据时态操作的时间字段查询矩阵的对应区域，获取到与操作时间有交错的索引的存储号。再根据存储号在索引表中查询到其数据及索引在 HDFS 中的储存地

址,Spark 集群逐一加载涉及到的数据集和索引至内存中执行时态操作,最后将结果集进行合并即为查询结果。如“查询时间为 2 005 至 3 078 各个时间点的数据库信息”,索引矩阵以 1 000 个时间版本为步长,边长 6 为例,则计算其开始结束的 hash 值为 2 和 3,根据 hash 值访问索引矩阵的对应区域(即蓝色阴影标注区域)得到有时间交错索引的存储号,即可检索对应索引表找到其 HDFS 地址。

跨段查询算法如算法 2 所示,其中 getHashCode(versionID)为计算时间的哈希值; serachIndexMartix(hashCode,indexMartix)为扫描索引矩阵,获取对应位置的索引块号。judgeTimeRegion(item,querytime,indexTable)为检索索引表,获取时间区域有重叠的索引块号。之后调用 readIndexFromHDFS(path)从 HDFS 中读取时态索引,进而进行时间旅行 temporalTravel(temporalIndex,versionID)得到查询结果。

算法 2　跨段查询

输入: versionID 时间版本,indexMartix 索引矩阵,indexTable 索引表;

输出: Result 查询结果集;

```
1. hashCode←getHashCode(versionID);
2. I←searchIndexMartix(hashCode);
3. For otem in I
4. P←judgeTimeRegion(item,versionID,indexTable);
5. End for
6. For path in P
7. temporalIndex←readIndexFromHDFS(path);
8. Result←temporalTravel(temporalIndex,versionID);
9. End for
```

3.4　索引更新

时态数据是带有时间属性的,这种特性也预示着时态数据会随着时间不断地增加,集群需要在新数据上添加索引以加快访问速度,并且集群无法将不断增大的数据及索引都维护在 Spark 集群中,需要一个索引的更新管理策略来控制 Spark 读取对应时间点或时间段所涉及的索引结构。时态数据的时效性特点导致了时态记录存在时间跨度且这种时间跨度是无法预料的,集群无法预知同一个对象所产生的下条记录的时间跨度长短。如果将新产生的时态记录索引到原有的索引结构中,需要在索引结构中进行频繁的队列移动,将新的索引节点插入到对应位置,并且这种插入式的方法无法批量操作,索引只允许逐一的进行插入,这种低效的更新策略是无法满足当今的查询需要的。

由于将新的时态数据索引到原有索引结构上这种索引更新策略所引起的时间开销要大于直接重构索引,本文拟采用滞后更新的索引更新策略。原有索引不去索引新的数据,而是当新的数据积累到一定量时,对新数据单独构建索引结构。

4　实验与结果

为了证明在 Spark 集群上构建时态索引的高效性,本节进行了以下几个方面的实验:(1) 在 Spark 集群环境下利用优化的 checkpoint 和原始 checkpoint 性能对比;(2) 在 Spark 环境下时态连接的性能效率;(3) 在 Spark 环境下分段索引性能分析。

4.1　实验环境

本次实验中 Spark 部分试验是在是由 5 台机器构成的 Spark on standalone 集群系统上完成的,1 个 master 节点和 4 个 slave 节点,各节点 CPU 为 Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz,6 GB 内存空间,采用 CentOS 6.8 系统,Scala 版本为 2.11.8,Spark 版本为 2.0.0。

4.2　实验数据集

本实验在 TPC-H 基准数据集的基础上扩展时态属性,并通过 TPC-C 事务生成基准数据集的历史数据[15]。本实验采用数据集如表 2 所示。

表 2　实验数据集

Dataset Size	SF_0	SF_H	10−6×|lineitem|	10−6×#version
Tiny	0.01	0.01	0.3	0.2
Small	0.1	0.1	3.4	2.2
Medium	1	1	34	22
Large	10	10	340	220

其中,SF_0 为 TPC-H 生成数据的比例因子;SF_H 为 TPC-C 生成数据的比例因子,即数据集中时间版本的规模;lineitem 表示数据集的总行数,version 表示时间版本规模。

4.3　实验结果与分析

4.3.1　checkpoint 优化前后性能对比

为了规避时态操作时分段查询干扰 checkpoint 性能判断的问题,本实验在小型数据集基础上对比 checkpoint 优化前后的性能变化,确保数据在一个数据段中。实验中查询给定 versionID 的数据库有效记录。实验结果如图 6 所示,图中横坐标为查询的 versionID。

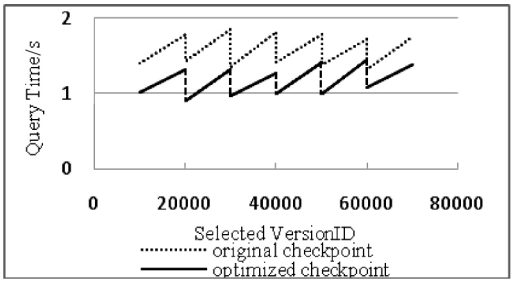


图 6 checkpoint 优化性能对比

图中查询时间随着查询时间版本的增长出现折现波动，这是因为 checkpoint 保存了对应时间版本下的数据库状态，无需遍历索引，而两个 checkpoint 中的时间版本数据库状态需要遍历时间线索引的对应区间，所以在 checkpoint 时间点上查询时间突然下降，查询两个 checkpoint 间的时间版本数据库状态查询时间是线性增长的。实验结果表明，优化后的 checkpoint 相对于原始 checkpoint 结构在时态旅行上提升 40% 左右。由于优化后的 checkpoint 将 visible rows 分段，并添加对应的标识位，减去了不必要的遍历的时间，这对于查询性能的提升是显著的。并且当数据集变大时，checkpoint 中需要保存的记录号变多了，则利用 checkpoint 进行时态操作时的遍历量变大，而优化的 checkpoint 规避了这种遍历所引起的时间开销，也就是说数据量越大时，优化的 checkpoint 对于性能的提升越明显。

4.3.2 跨区查询时间开销

实验结果如图 7 所示，优化后连接算法的执行效率要明显由于原有连接算法。进行连接优化后，时间开销随连接记录数增加呈线性增长，而未优化的连接算法呈几乎指数级的增长态势。在连接量较少时，原有连接算法略优于优化算法，这是因为数据混洗引起的时间开销高于优化的时间代价。而连接的记录数越多，优化的连接算法优势越大，其原因主要在于未优化连接操作低效地两两连接，而优化后连接算法通过 hash 成功规避了低效连接操作，缩小了连接范围。

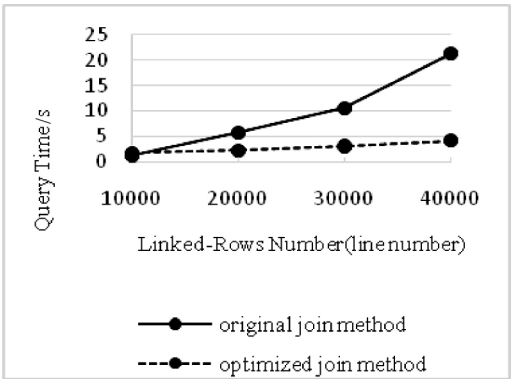


图 7 跨区查询性能对比

4.3.3 跨段查询时间开销

实验结果如图 8 所示，在四次不同时间版本跨度的查询中，有三次查询的耗时在 5 秒左右，只有最后一次查询的时间开销为 10 秒，这是因为前三次范围查询只涉及一个数据段，最后一次范围查询涉及到了两个数据段。Spark 集群将时态数据分段构建索引，在进行时态操作时，集群首先查询全局索引，将时态区间与查询时间版本有交叉的索引文件逐一从 HDFS 读入 Spark 集群进行查询。同时，由实验数据可知，在进行范围查询时往往只涉及到一至两个数据段，这是因为数据本身的时态特性就决定了数据本身在时间维度上是聚集连续的，在进行时态范围查询时只涉及到少数的数据段。因此，分段查询在解决了数据量过大无法构建索引问题的同时，还保证了查询的效率。

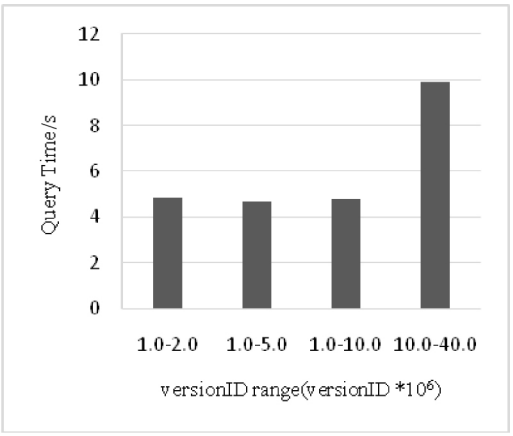


图 8 跨段查询时间开销

5 结 语

本文针对现有时态索引在数据量过大，普通硬件环境难以支持的问题，基于时态数据的特性，提出基于 Spark 的分布式时态索引方法。该方法在 Spark 集群上按数据分区单独构建时态索引，在此基础上优化辅助索引 checkpoint 结构，设计面向对象维度的辅助索引以及全局双 hash 结构，全面高效地支持各种时态查询操作。最后，通过基准数据集上的实验验证了所提方法的有效性。目前本文所实现的 Spark 时间索引是在离线数据的基础上，在接下来的研究中，我们准备实现实时监控的流式构建时态索引系统，并进一步优化在跨区查询时面对大量混洗数据的查询算法。

参 考 文 献

[1] Yong Tang. TimeDB-A Temporal Relational DBMS [OL]. [2015-03-05]. http://www.timeconsult.com/software/software.html, 1999.

(下转第 163 页)

参 考 文 献

[1] 张少中，方朝曦，陈军敢，等. 基于社会网络的电子商务信任社区聚类模型[J]. 浙江大学学报(工学版)，2013，47(4):656－661.

[2] Gregory S. An Algorithm to Find Overlapping Community Structure in Networks[C]// European Conference on Principles and Practice of Knowledge Discovery in Databases. Springer-Verlag，2007:91－102.

[3] Lancichinetti A，Fortunato S，Kertész J. Detecting the overlapping and hierarchical community structure of complex networks[J]. New Journal of Physics，2008，11(3):19－44.

[4] Palla G，Derényi I，Farkas I，et al. Uncovering the overlapping community structure of complex networks in nature and society[J]. Nature，2005，435(7043):814－818.

[5] Ahn Y Y，Bagrow J P，Lehmann S. Link communities reveal multiscale complexity in networks[J]. Nature，2009，466(7307):761－764.

[6] Evans T S，Lambiotte R. Line graphs，link partitions，and overlapping communities[J]. Physical Review E Statistical Nonlinear & Soft Matter Physics，2009，80(1Pt2):016105.

[7] Kim Y，Jeong H. Map equation for link communities[J]. Physical Review E Statistical Nonlinear & Soft Matter Physics，2011，84(2):026110.

[8] Krackhardt D. Structural Holes: The Social Structure of Competition[J]. Administrative Science Quarterly，1995.

[9] Girvan M，Newman M E. Community structure in social and biological networks[J]. Proceedings of the National Academy of Sciences of the United States of America，2002，99(12):7821.

[10] Pan L，Dai C，Wang C，et al. Overlapping Community Detection via Leader-Based Local Expansion in Social Networks[C]// IEEE，International Conference on TOOLS with Artificial Intelligence. IEEE，2012:397－404.

[11] Chen D，Shang M，Lv Z，et al. Detecting overlapping communities of weighted networks via a local algorithm[J]. Physica A Statistical Mechanics & Its Applications，2010，389(19):4177－4187.

[12] Lee C，Reid F，Mcdaid A，et al. Detecting highly overlapping community structure by greedy clique expansion[C]// Proc of the 4th Int Workshop on Social Network Mining and Analysis(SNNKDD'10)，New York:ACM，2010:33－42.

(上接第 108 页)

[2] Saracco C M，Nicola M，Gandhi L. A matter of time:Temporal data management in DB2 10[R]. Technical report，IBM，2012.

[3] Kaufmann M，Fischer P M，May N，et al. TPC-BiH:A Benchmark for Bitemporal Databases[M]//Performance Characterization and Benchmarking. Springer International Publishing，2013:16-31.

[4] Elmasri R，Wuu G T J，Kim Y J. The time index:An access structure for temporal data[C]//Proceedings of the 16th International Conference on Very Large Data Bases. San Francisco，CA:Morgan Kaufmann Publishers，1990:1-12.

[5] Tao Y，Papadias D. Efficient historical R-trees[C]//Scientific and Statistical Database Management，2001. SSDBM 2001. Proceedings. Thirteenth International Conference on. IEEE，2001:223-232.

[6] Zhang F，Wang X，Ma S. Temporal XML Indexing Based on Suffix Tree[C]//Acis International Conference on Software Engineering Research，Management and Applications. IEEE，2009:140-144.

[7] Kaufmann M，Manjili A A，Vagenas P，et al. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA[C]//ACM SIGMOD International Conference on Management of Data. ACM，2013:1173-1184.

[8] Kaufmann M，Fischer P M，May N，et al. Bi-temporal Timeline Index:A data structure for Processing Queries on bi-temporal data[C]//IEEE，International Conference on Data Engineering. IEEE，2015:471-482.

[9] Zaharia M，Chowdhury M，Franklin M J，et al. Spark:cluster computing with working sets[C]//Usenix Conference on Hot Topics in Cloud Computing. USENIX Association，2010:10-10.

[10] Becker B，Gschwind S，Ohler T，et al. An asymptotically optimal multiversion B-tree[J]. The VLDB Journal，1996，5(4):264-275.

[11] Tao Y，Papadias D. MV3R-Tree:A Spatio-Temporal Access Method for Timestamp and Interval Queries[C]//VLDB 2001，Proceedings of，International Conference on Very Large Data Bases，September 11-14，2001，Roma，Italy. DBLP，2001:431-440.

[12] Liao H，Han J，Fang J. Multi-dimensional Index on Hadoop Distributed File System[C]//International Conference on Networking，Architecture，and Storage，Nas 2010，Macau，China，July. 2010:240-249.

[13] Gankidi V R，Teletia N，Patel J M，et al. Indexing HDFS data in PDW[J]. Proceedings of the VLDB Endowment，2014，7(13):1520-1528.

[14] Xie D，Li F，Yao B，et al. Simba:Efficient In-Memory Spatial Analytics[C]//International Conference on Management of Data. ACM，2016:1071-1085.

[15] Funke F，Kemper A，Krompass S，et al. Metrics for measuring the performance of the mixed workload CH-benCHmark[C]//Tpc Technology Conference on Topics in PERFORMANCE Evaluation，Measurement and Characterization. Springer-Verlag，2011:10-30.