第46卷第1期 中国科学技术大学学报 Vol. 46, No. 1
2016年1月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jan. 2016

文章编号:0253-2778(2016)1-0066-10

基于 Spark/Shark 的电力用采大数据 OLAP 分析系统

王亚玲[1],刘 越[2],洪建光[3],崔 蔚[1],李彦虎[2],苏伊鹏[2],黄高攀[4],张明明[4],刘万涛[2]

(1. 国网信息通信产业集团有限公司,北京 100761;2. 中国科学院计算技术研究所,北京 100190;
3. 国网浙江省电力公司,浙江杭州 310007;4. 国网江苏省电力公司信息通信分公司,江苏南京 210029)

摘要:用电信息大数据上的 OLAP 查询涉及数据量大,具有多表连接操作频繁、SQL 结构复杂等特点,传统关系型数据库面对该类应用,表现出可扩展性弱、数据写入吞吐量低与查询效率低等问题. 为此设计了一套基于 Spark/Shark 的电力大数据 OLAP 分析系统,该系统采用分布式文件系统 HDFS 保存电力用电信息采集系统的大数据,通过 Shark 进行前端 SQL 解析,Spark 进行查询计算;然而,原生 Shark 只支持粗粒度分区,不支持细粒度的索引技术,难以高效地过滤无关数据,影响了查询性能. 为克服这一不足,该系统设计了一种基于前缀树的细粒度索引结构 TrieIndex,并通过数据重组技术优化了数据在 HDFS 的分布,提升了 Shark 的数据过滤能力以及用电信息大数据 OLAP 分析的性能. 真实用电信息采集系统数据与查询的实验结果表明,该系统比关系型数据库的写入速度提升了 12 倍,比原生 Shark 的查询效率提升了 10 倍以上.

关键词:Spark;OLAP;电力大数据;索引;前缀树

中图分类号:TP18 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2016.01.009

引用格式:WANG Yaling, LIU Yue, HONG Jianguang, et al. Spark/shark-based OLAP system for smart grid applications[J]. Journal of University of Science and Technology of China, 2016,46(1):66-75.
王亚玲,刘 越,洪建光,等. 基于 Spark/Shark 的电力用采大数据 OLAP 分析系统[J]. 中国科学技术大学学报,2016,46(1):66-75.

Spark/Shark-based OLAP system for smart grid applications

WANG Yaling[1], LIU Yue[2], HONG Jianguang[3], CUI Wei[1]
LI Yanhu[2], SU Yipeng[2], HUANG Gaopan[4], ZHANG Mingming[4], LIU Wantao[2]

(1. *State Grid Information & Telecommunication Group Co. Ltd.*, *Beijing* 100761, *China*;
2. *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*;
3. *State Grid Zhejiang Electric Power Company*, *Hangzhou* 310007, *China*;
4. *State Grid Jiangsu Electric Power Company Information & Telecommunication branch*, *Nanjing* 210029, *China*)

Abstract: The OLAP queries on electricity consumption information in Smart Grid have some prominent features: huge amounts of data, involving multiple tables in a joint operation, complex SQL structure, etc. Faced with this kind of applications, traditional RDBMS always leads to poor scalability, low write throughput, and unacceptable query performance, etc. A Spark/Shark-Based OLAP system for electricity consumption information in smart grid was designed. The system used distributed file system HDFS for data storage, and makes use of Shark to parse the SQL queries and Spark to execute them. However,

收稿日期:2015-08-27;**修回日期**:2015-09-29
基金项目:国家电网公司科技项目(SGJSXT00YWJS1400072)资助.
作者简介:王亚玲,女,1972 年生,硕士/高级工程师. 研究方向:数据挖掘. E-mail: wangyaling@sgitg.sgcc.com.cn
通讯作者:刘万涛,博士/助理研究员. E-mail: liuwantao@ict.ac.cn

Shark does not support fine-grained index, which hinders further improvement of query performance. To overcome this limitation, a Trie tree based fine-grained index technique TrieIndex and data re-organization scheme for better query performance was proposed. The experiment results with real electricity consumption information data and query show that the write throughput of the system is 12 times faster than that of RDBMS, and the query efficiency of the system is 10 times greater than that of original Shark.
Key words: Spark; OLAP; power big data; index; Trie tree

0　引言

智能电网是近年来国际上电网技术的主要发展方向. 与传统电网不同,智能电网在发电、输电、配电、用电等电力生产与消费的全生命周期各环节有机融合了 IT、自动化、网络等技术,全面提升了电网的"感知、分析、响应"水平. 通过在电网的神经末梢部署大量的量测装置(如智能电表、多功能电子表、各类传感器等),实时采集并分析各类数据、动态调配需求与供给,从而保证电网平稳运行、提高能源使用效率、降低能源损耗. 近年来,随着相关技术的逐步成熟,我国智能电网建设的规模不断扩大.

用电信息采集(采用)系统是智能电网的重要组成部分. 该系统通过部署在用户侧的各类量测装置将用户用电量、电压、电流等多项数据进行编码后通过网络发送给电网公司的数据中心进行统一存储和分析处理,并支撑电量计算、窃电分析、线损分析等各类上层用采应用.

传统的用采数据 OLAP 分析系统依赖于关系数据库架构,部署于小型机集群上,由于传统电表通常每月采集、上报一次数据,数据量小、数据项少、频率低,因此这种架构能够很好地满足用采系统对数据存储与处理的需求.

由于智能电表的数据采集、上报频率显著高于传统电表,因此国家电网规范要求智能电表每 15 min 采集并上报一次数据(每天 96 次),未来还有可能增至每 5 min 一次(每天 288 次),一个省就会部署多达数千万块智能电表. 除数据采集频率极高外,智能电表采集的数据内容丰富,包含 30 多个数据项. 一方面为更深入地分析数据提供了便利与可能性,另一方面也使得用采系统的数据总量与数据复杂程度急剧膨胀.

传统的基于关系数据库的用采数据 OLAP 分析系统难以满足对智能电表产生的海量数据进行有效分析和查询,暴露出以下三个突出问题:

(Ⅰ)数据写入性能低下. 关系数据库通常采用索引优化查询性能,而海量用采数据频繁的插入、更新操作,致使索引维护的成本极高,造成数据写入性能低下.

(Ⅱ)大数据场景下多表连接性能低下. 用采数据的 OLAP 查询往往涉及报表数据(或上报数据)与用户信息、设备信息等档案类数据的多表连接操作,随着数据量的急剧增加,这类操作的执行性能迅速降低.

(Ⅲ)扩展性差. 关系型数据库的规模无法随着采集数据量以及上层业务场景的增加而灵活地扩展,阻碍了业务系统的快速发展.

近几年,以 Hadoop[1]、Spark[2-3] 为代表的分布式存储与计算框架在工业界得到了广泛的应用,它们具有高写入吞吐能力、灵活的可扩展性与高可用性等特点. Hadoop 中的高吞吐分布式文件系统 HDFS 可以为电力大数据 OLAP 系统提供稳定可靠的海量数据存储能力,而 Spark[4] 可以为其提供高性能的内存计算框架. 此外,Spark 还提供了 SQL 兼容的数据仓库系统 Shark,为分析人员提供了便利的查询工具.

为解决上述问题,本文设计并实现了一套基于 Spark/Shark 的电力用采大数据 OLAP 分析系统,通过分布式存储与计算技术提升查询性能与系统可扩展性. 此外,用采数据上的 OLAP 查询往往涉及区域层级关系的细粒度数据过滤条件,而 Shark 只支持粗粒度的分区操作,不支持细粒度的索引,无法提供高性能的 OLAP 查询处理能力. 针对这一局限性,本文提出了一种基于前缀树的分布式索引结构 TrieIndex,并通过数据重组织技术优化了数据在 HDFS 的分布. 该索引结构可建立在粗粒度的分区之上,提供细粒度的数据索引功能,两者互补,通过更高效的数据过滤,极大地提高了 Shark 的查询处理速度.

1　相关工作

在 Hadoop 上的索引研究方面,文献[5]提出了

一种分布式数据库系统 HadoopDB，该系统结合了 Hadoop 与 RDBMS 的优点，使用 Hadoop 的 MapReduce 框架与 RDBMS 的索引技术来处理大规模数据. 文献[6]提出了一种简单的一维区间索引，该索引可以加速 HDFS 上排序文件的读取速度. 文献[7]提出了 Trojan Index 和 Trojan Join Index 两种索引结构. 第一种索引通过记录每个数据片的最小值和最大值来过滤无关的数据片，第二种索引通过把未来要进行 Join 的表进行统一存储来提高 Join 查询的效率. 文献[8]提出的索引结构可以记录每个数据片中各数值维度和日期维度的最小值和最大值，并为字符串类型维度建立倒排索引来提升查询速度. 以上的工作只能过滤较粗粒度的数据片，这会导致读取过多冗余的无关数据，降低 OLAP 的查询性能. 此外，文献[9]提出了一种 Hive 上的 DGFIndex 多维索引结构. 该索引可以在细粒度上过滤无关数据，但无法表达区域层级的树形关系. 本文中的 TrieIndex 使用了与 DGFIndex 类似的建立索引方式与索引保存方式.

在电力大数据处理方面，文献[10]针对目前用采系统的存储瓶颈，基于云存储文件系统，设计了通用的数据接口，为上层应用提供数据支撑. 文献[11]在分析大数据、云计算与智能电网三者关系的基础上，提出电力大数据平台的总体架构以及四项关键技术：集成管理技术、数据分析技术、数据处理技术与数据展现技术，并给出了三个电力大数据应用的典型案例.

2 系统设计与实现

2.1 系统架构

图 1 展示了用采数据分析系统的架构图，整个系统由数据接入层、用采系统、ETL 工具以及关系数据库等组件构成.

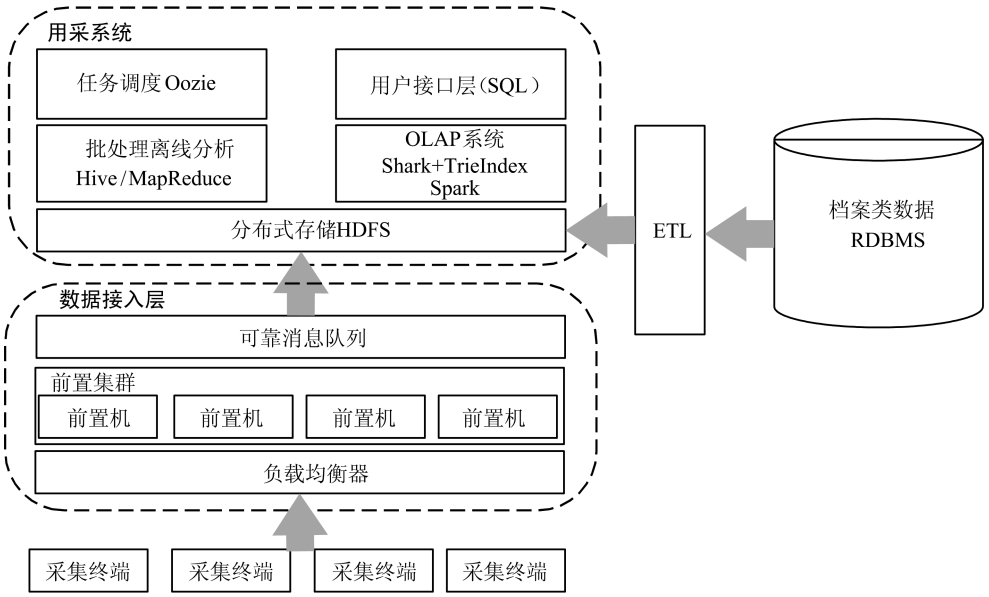

图 1 用采数据分析系统架构图

Fig. 1 Architecture of the smart electricity consumption information system

数量众多的采集终端以固定的频率采集用电数据，通过网络把数据发送至数据接入层. 数据接入层由负载均衡器、前置集群和可靠消息队列等组件构成. 由于采集类上报数据频率高、数量大，因此需要经由负载均衡器接收上报数据，并按照一定的规则分发至前置集群对数据进行解包与规约等处理，产生符合业务规范要求的统一格式的数据. 为避免规约解析后的上报数据产生的速度与分布式存储的速度不匹配，将上报数据导入分布式可靠消息队列，该组件可实现高频海量数据的接收、缓冲和可靠消息分发功能. 分布式存储系统采取“拉”的方式，从可靠消息队列中取出数据并进行持久化存储. 高吞吐的 HDFS 可满足高频采集类数据的写入需求.

除采集终端上报的数据外，用采数据分析还需要用到用户信息、电力设备(智能电表、变压器等)信息等档案类数据. 由于档案类数据修改较为频繁，且被多种采用关系数据库架构的业务系统共享，因此使用 RDBMS 维护. 用采系统通过 ETL 工具将需要

的档案类数据导入 HDFS.

用采数据分析分为离线分析与 OLAP 分析两类. 离线分析包括线损分析、数据完整率计算、电量计算、异常分析等业务场景，其特点是业务逻辑复杂、运行时间较长(数十分钟至数小时). 为避免资源竞争，通常在夜间非工作时间对前一天的用采数据进行离线分析. OLAP 分析则是指前端业务系统发出的各种 Ad-Hoc 查询，通常需要把离线分析生成的报表数据或上报数据与档案类数据进行 join 操作，如查询某市的耗电量最多的 10 个用户的抄表数据、查询某个电力局专变用户当前已装终端明细等. 具有分析任务不固定、响应时间要求较高(数秒钟至数分钟)等特点. 具体系统设计上，离线分析采用 Hive[12] 与 MapReduce[13] 计算框架，由 Oozie[14] 进行任务调度，产生的报表数据写回 HDFS，然后交由 OLAP 系统进行在线分析查询使用. OLAP 分析采用内存计算框架 Spark 及其上的 SQL 工具 Shark. 由于采用了分布式内存计算技术，Spark 适用于对海量数据进行响应时间要求较为苛刻的各类分析计算；而 Shark 则在 Spark 之上提供了一套类 SQL 接口，降低了用户的使用难度，并缩短了现有基于关系数据库的业务逻辑的迁移过程. 针对用采数据特征以及 Shark 不支持细粒度索引的局限性，本文设计并实现了一种基于前缀树的细粒度索引结构 TrieIndex，进一步提升了用采数据 OLAP 分析的性能.

2.2　Spark 与 Shark

Spark 以 resilient distributed datasets(RDD)形式的数据抽象模型作为分布式计算阶段间的数据交换方式，RDD 是一系列数据片的集合，可以缓存在内存中. 相较 Hadoop 使用磁盘交换数据的方式，极大地提升了计算的性能. 此外，Spark 提供了比 Hadoop 更丰富的操作类型，适用于交互式数据处理、批处理、流处理等多种应用类型.

Shark 是 Spark 之上的类数据仓库工具，并提供了 SQL 接口. Shark 可以将每条 SQL 查询转换为分布式查询计划，而该计划可以在 Spark 中运行. 在 Shark 中，一张表即为 HDFS 中的一个目录，该目录下的文件即为表中的数据. 此外，Shark 支持分区，一个分区为表目录下的一个子目录，这样查询可以通过指定分区在较粗粒度上过滤掉查询无关的数据.

2.3　OLAP 查询特点

电力用采大数据上的 OLAP 查询具有如下特点：①多表连接操作频繁，大量查询表现为报表数据与多个档案类数据表的连接或采集类数据表与多个档案类数据表的连接，并且连接操作涉及表的数量较大，一般为 5～8 张表. ②数据量庞大，智能电表每 15 min 采集并上报一次数据(每天 96 次)，一个省的部署量即高达数千万块，每天新增数十亿条采集类数据. 此外，由于用电客户数量大、各类电力设备种类、型号众多，使得各省的档案表极为庞大，如南方某省的用户信息包含数千万条记录. ③查询中每个表都有对区域层级和日期的过滤条件(区域层级的过滤操作包括 LIKE、=等)，并且过滤粒度各异. 而用采数据的区域层级编码呈现出前缀树结构. 例如，某省的区域层级编码为 101，该省某市的编码为 10101，该市某个区的编码为 1010101. 这样，全省的区域层级编码构成一棵前缀树. 此外，从省级一直到最细粒度的区域(供电所)具有多个层次，且随着层次的深入，每层的节点数快速增加，从而使得该前缀树结构比较庞大.

在原生 Shark 处理查询时，需要对查询中涉及的每个表进行顺序全表扫描，找到所需的数据记录. 如前所述，用采 OLAP 查询具有多表连接操作频繁与数据量庞大两个特点，从而使得顺序扫描需要耗费大量的资源与时间，效率低下.

Shark 的分区机制能够将一张数据表按照某个属性的值划分为多个分区，每个分区对应一个单独的目录，查询时通过指定分区过滤掉无关数据，达到提高查询性能的目的.

对于用采 OLAP 分析而言，如果在较大粒度区域层级(如市级)建立分区，则难以充分过滤和查询无关的数据，继而难以保证查询的响应时间. 如果在用采数据的最细粒度区域层级建立分区，则会造成分区数量过多. 由于 HDFS 将目录结构和文件信息都保存在主节点的内存中，过多的分区数(即目录数)将会给 HDFS 的主节点造成很大的压力. 此外，Shark 处理查询前，需要首先读取相关表的分区元信息，这些数据往往保存在 RDBMS 中. 如果分区数量过多，则会导致 Shark 需要读取大量元数据从而降低查询启动速度.

2.4　TrieIndex

为解决 Shark 不支持细粒度索引的问题，本文提出一种基于 Trie 树的一维索引结构 TrieIndex，

支持按照用采数据的最细粒度区域层级构建索引，查询时通过索引过滤无关数据，提高查询性能；并通过数据重组织技术，将具有相同索引值的数据记录连续存储，提升了数据顺序读取效率，也进一步提高了查询性能.

Trie 树利用字符串的公共前缀提高查询、检索性能，将根节点到某一结点途经的节点连接起来，即为该节点对应的字符串. 具体到用采数据，Trie 树的每一层对应区域层级编码的一个层级，叶节点表示最细粒度的层级编码. 在叶节点处建立索引，即叶节点通过指针指向所有以该节点的值作为区域层级编码的记录.

图 2 所示为一个简化的用采数据区域层级编码对应的 Trie 树模型，除根节点外，该树的三层分别对应市级、区级、供电所级的层级编码. 图 2 中，值为 1010101 的叶节点指向所有区域层级编码为 1010101 的记录. TrieIndex 中的 Trie 树结构通过读取外部表构建，保存在内存中以提高访问速度. 而叶节点与数据片位置的映射关系表保存在 HBase 中，这是因为叶节点要保存每个建立索引的表的数据片的位置关系，加之用采系统中表的数量众多，导致无法将完整的 Trie 树与映射关系保存在内存中. 此外，如果内存中的 Trie 树结构过大，会造成系统启动过慢、故障恢复过慢等问题. HBase 中表的记录为键值形式，键表示最细粒度区域层级编码，值表示具有该编码的所有记录所在的文件名和偏移量.

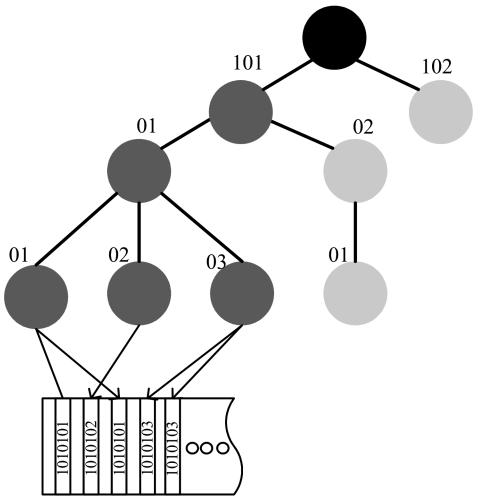

图 2 TrieIndex 结构

Fig. 2 Structure of the TrieIndex

具有同一个区域层级编码的记录通常随机地散落在数据文件的不同位置，图 3 是重组前的数据文件所示. 在分布式文件系统 HDFS 中，这些记录可能会位于不同的数据块中，从而存储在不同的节点上. 当 Shark 通过 Trie 树索引检索某一区域层级编码时，会产生大量的随机磁盘读取操作，进而因 Trie 树索引影响性能提升.

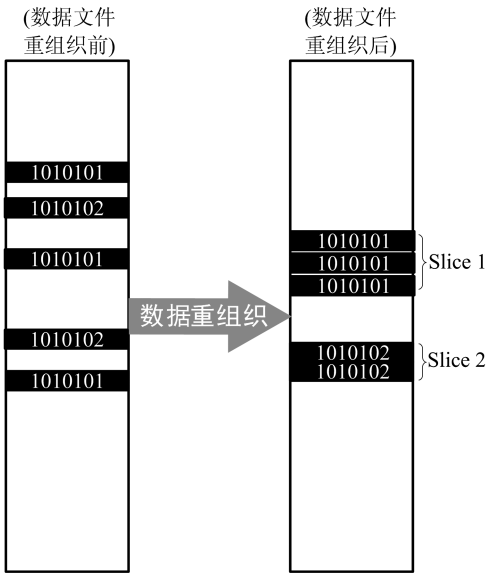

图 3 数据重组织示意图

Fig. 3 Data re-organization

为解决这些问题，本文设计了一种数据重组织技术，以使所有具有相同区域编码的记录连续存储，这样可将低效的磁盘随机读取操作转换为高效的顺序读取操作. 这些记录所在的数据片被称为一个 Slice，如图 3 所示.

2.4.1 索引建立

TrieIndex 的建立分两个步骤. 第一步创建 Trie 树；第二步数据重组织并建立叶节点映射关系的过程.

第一步，如算法 2.1 所示，Shark 首先从 RDBMS 中读取保存区域编码的表（第 1 行），并声明 Trie 树为空（第 2 行）；然后用读取的所有区域编码创建 Trie 树（第 3～5 行）；最后返回创建好的 Trie 树（第 6 行）. Trie 树构建完成后，该结构常驻 Shark 所在服务器的内存中，这样可以加速处理查询时访问索引的速度.

算法 2.1 Trie 树建立过程.

输入：区域层级关系的外部表（保存在 RDBMS 中）

输出：Trie 树

```
1 regionCodeSet = readFromRDBMS();
2 TrieTree = null;
3 FOR regionCode in regionCodeSet DO;
4 TrieTree.insert(regionCode);
```

5 END FOR;

6 return TrieTree.

第二步,首先对数据表按照日期分区,然后在每个分区上建立 TrieIndex. 建立索引并不会增加表下的目录数,只是把数据文件内的记录重新组织一下,即改变它们的相对顺序. 由于用采数据量巨大,因此建立索引的过程为一个 MapReduce 任务,在多个节点上并行进行,该任务的流程如图 4 所示. 由于建立索引的过程是对全表数据重组,因此建立索引的 MapReduce 不需要使用 Combine 函数,因为其并不能减少需要写出的数据量.

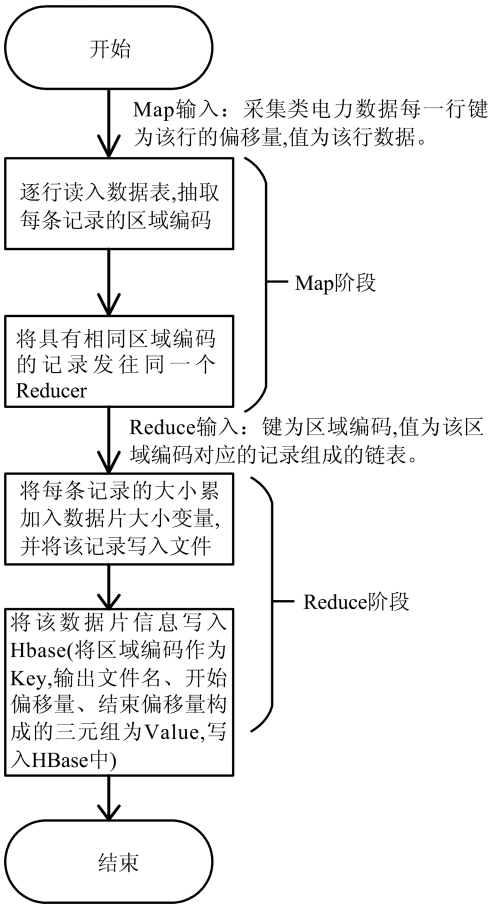

图 4　数据重组织 MapReduce 过程流程图

Fig. 4　The MapReduce flow chart for data re-organization

Map 阶段的逻辑如算法 2.2 所示,在 Map 函数中,对每条记录使用区域编码的列号(在不同的表中,区域编码的列号不同,通过元数据可以得到该信息,第 1 行)抽取该记录的区域编码字段的值(第 2 行),以该值为 Key,整条记录为 Value,发送给 Reduce 函数(第 3 行).

算法 2.2　TrieIndex 建立过程 Map 函数.

输入:采集类电力数据每一行:键为该行的偏移量,值为该行 line

输出:键为区域编码,值为整条记录

1 idx = conf. get(REGION_CODE_COL);

2 key = getRegionCode(line, idx);

3 Emit(key, line).

Reduce 阶段利用 Map 阶段的输出完成数据重组织,其逻辑如算法 2.3 所示. 在 Reduce 函数里,记录当前输出文件的偏移,即开始偏移量(第 1 行),初始化 Slice 的结束偏移量为−1,当前 Slice 的大小为 0,得到当前输出文件的名字(第 2～4 行). 对某个 Key(即区域编码)而言,遍历具有该 Key 的所有记录,对某条记录而言,计算该记录的长度,累加到 Slice 大小变量中;然后直接输出该记录到输出文件中(第 5～8 行). 接下来,计算结束偏移量的值,即开始偏移量与 Slice 大小之和(第 9 行);最后将区域编码作为 Key,输出文件名、开始偏移量、结束偏移量构成的三元组 Value,写入 HBase 中. 这样,经过索引建立的过程,最细粒度的区域编码与对应数据片(Slice)之间的关系被保存在 HBase 中的索引表中.

算法 2.3　TrieIndex 建立过程 Reduce 函数.

输入:键为区域编码 region、值为该区域编码对应的记录链表 lineList

输出:索引信息写入 HBase,整条记录输出到文件

1　start = 输出文件的当前偏移;

2　end = −1;

3　sliceSize = 0;

4　filename = 输出文件的名字;

5　FOR line in lineList DO

6　sliceSize += sizeOf(line);

7　output(line);

8　END FOR

9　end += sliceSize;

10 HBase. put(region,<filename,start, end>);

如图 5 所示,经过 TrieIndex 的创建过程,具有相同区域码的记录被重组织存储在一起,如 1010103 区域码的两条记录所示. 此外,在 Reduce 阶段,区域码与对应记录在文件中的位置信息也被写入 HBase 中. 该例中,假设每条记录的长度为 21 个字节,那么具有 1010103 区域码的两条记录在文件中的开始偏移和结束偏移分别为 148 和 189.

2.4.2　索引查询

Shark 使用 TrieIndex 查询的过程可以分为三个阶段:

(Ⅰ)如算法 2.4 所示. Shark 首先解析查询,得到查询谓词中区域编码相关的条件(第 1 行). 然后

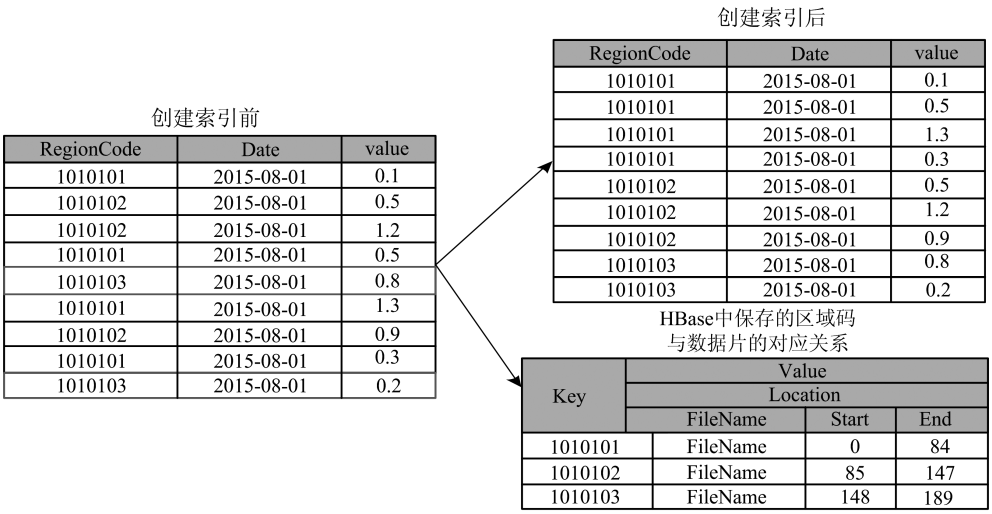

图 5 创建索引示意图

Fig. 5 Index creation

根据得到的条件查询内存中的 Trie 树，得到查询相关的最细粒度区域编码集合(第 2 行). 最后从 HBase 读取区域编码相关 Slice 的位置信息(第 3 行)，并写入 HDFS 中的临时文件以供后面的过程使用(第 4 行).

算法 2.4 GetQueryRelatedRegionCodes(Q)

输入：查询 Q

输出：最细粒度的查询相关的区域编码集合.

```
1 regionCodePred = extract(Q);
2 keyset = Trie.search(regionCodePred);
3 sliceLocations = HBase.getAll(keyset);
4 writeToTmpFileonHDFS(sliceLocations).
```

(Ⅱ)如算法 2.5 所示. 首先将过滤后得到的 Split(即 MapReduce 中的 InputSplit)集合变量初始化为空(第 1 行). 然后从输入表得到所有的 Split(第 2 行)，从算法 2.3 产生的临时文件得到与查询相关的各个 Slice 在文件中的位置(第 3 行). 再过滤那些与查询相关 Slice 不相交的 Split(第 4～8 行). 最后每个选中的 Split 得到其所有需要读取的 Slice 的位置(第 9～10 行)，并按照 Slice 的位置偏移从小到达排序(第 11 行)；使用 Split 的文件名字与 Split 偏移量作为键，所有在该 Split 内的 Slice 偏移列表为值，存入 HBase 的一张临时表中，以供后面过滤无关 Slice 时使用(第 12 行)；最终返回选中的 Split 集合(第 14 行).

算法 2.5 FilterUnrelatedSplits

输入：算法 2.3 得到的临时文件，里面包含查询相关 Slice 的位置信息.

输出：经过过滤后的 Splits，即 Spark 程序的输入 Splits.

```
1  finalSplits = Φ;
2  allSplits = getAllSplitsFromInputPath();
3  sliceLocations = readFromTmpFileOnHDFS();
4  FOR split in allSplits DO;
5  IF split.overlap(sliceLocations) THEN;
6  finalSplits.add(split);
7  END IF;
8  END FOR;
9  FOR split in finalSplits DO;
10 slicesLocs = getSlicesInSplit(sliceLocations);
11 Sort(slicesLocs);
12 HBase.put(split.name_split.start, slicesLocs);
13 END FOR;
14 Return finalSplits.
```

(Ⅲ)如算法 2.6 所示. RecordReader 负责实际的数据读取操作，该类首先从 HBase 中使用当前 Split 的名字和偏移量作为 Key 读取该 Split 中需要读取的 Slice 集合的偏移信息(第 1 行)，并读取每个 Slice 中的数据，发送给 Spark 的操作符，并过滤无关的数据(第 2～4 行).

算法 2.6 FilterUnrelatedSlices.

输入：算法 4 得到的 HBase 中的临时表、Split

输出：读取查询相关的 Slice，发送给 Spark

```
1 slicesLocs = HBase.get(split.name_split.start);
2 FORslice in slicesLocs DO;
3 sendDataToSpark(slice);
4 END FOR;
```

假设查询为：

```
SELECT sum(value)
FROM table
WHEREregioncode='1010103'
```

使用 TrieIndex 处理该查询的过程如图 6 所示. 首先 Shark 先在 Trie 树中找到对应的叶节点，其区域码为 1010103，使用该区域码查询 HBase 中的索引表，得到相关数据的位置，并写入 HDFS 上的一个临时文件中；然后 Shark 读取临时文件中的 Slice 位置信息，判断哪些 Split 与其相交，选定候选需要处理的 Split，并且为每个 Split 创建一个键值，键为 Split 的所在文件的名字和偏移量，值为其内所有需要读取的 Slice 的开始和结束偏移量排序后的列表. 该键值对会被写入到 HBase 的一张临时表中. 因为在本例中的文件较小，所以只有一个 Split，该 Split 中只有一个 Slice 需要读取；最后在读取该 Split 数据时，根据 Split 的键从 HBase 得到该 Split 中需要读取 Slice 的位置信息，在读取时，跳过无需读取的部分，直接读取查询相关的 Slice，最终得到计算结果.

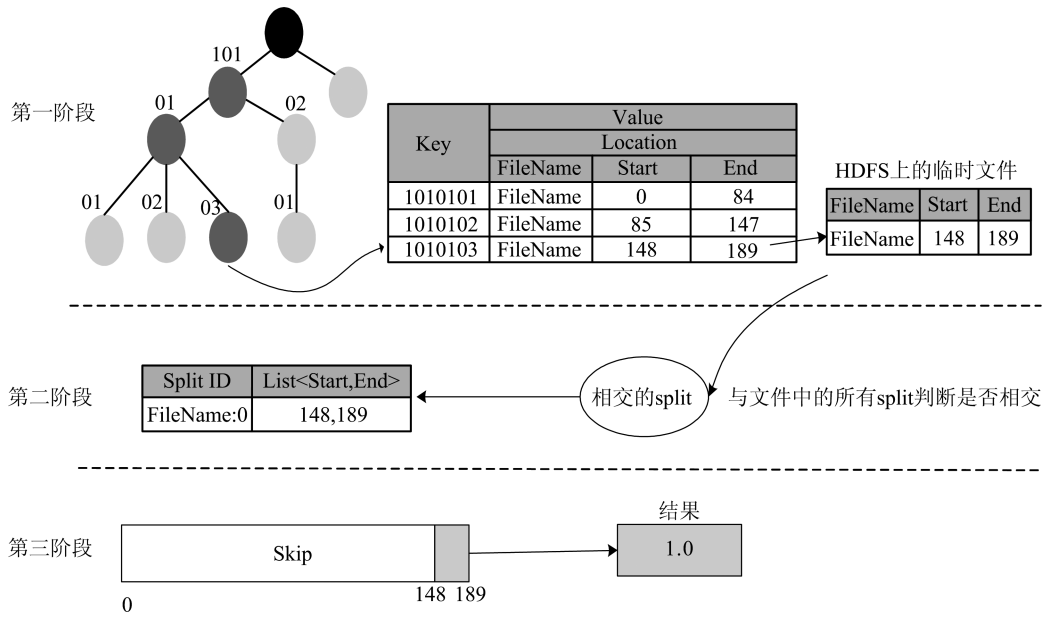

图 6　TrieIndex 查询过程示意图

Fig. 6　Illustration of a query using TrieIndex

TrieIndex 并没有像分区一样增加过多的目录，通过数据文件重组织达到了使用最细粒度的区域编码过滤无关数据的目的，极大地提高了查询性能.

2.5　系统实现

本系统可以通过命令行或 JDBC 连接的方式使用 SQL 的 Create Index 命令创建索引，该命令经过 Shark 解析得到数据重组织相关的参数，用 Hadoop 的客户端提交一个 MapReduce 程序进行索引创建.

查询时，Shark 会把 SQL 查询解析为语法树，通过检查语法树中的谓词是否包含区域层级编码条件来决定是否使用索引. 数据的过滤读取主要是通过定制化 Hadoop 的 InputFormat 和 RecordReader 实现. 具体而言，在 Hadoop 中，InputFormat 的 getSplits 函数负责从数据输入路径得到 Mapper 需要处理的 Split(一般为 64 MB). 本系统实现了基于 TrieIndex 的 InputFormat 类，该类使用算法 2.4 和 2.5 得到查询相关的 Splits，这一步是粗粒度的数据过滤. 此外，在 Hadoop 中，Mapper 通过迭代调用 RecordReader 类的 next 函数得到每条记录，然后才能做后续处理. 本系统定制化实现了 RecordReader 类，该类的 next 函数使用算法 2.6 进行 Split 内的无关数据过滤，这一步是细粒度的数据过滤. 因为本系统只修改了 Shark 中与索引建立和查询相关的代码，并未对 HDFS 和 Spark 造成影响，因此可以充分利用这两者的可扩展性、高可用性等优点.

3　实验设计与结果

本实验的目的是验证基于 Spark/Shark 的电力大数据 OLAP 系统的性能，与关系型数据库及原生

Shark进行对比.本实验使用由28个虚拟机节点组成的集群,每个节点具有8核CPU,8 GB内存.此外,实验采用的数据集与查询集均来自真实的电力业务系统,包含7张表,7亿条数据,数据总规模为200 GB.每个实验重复3次,并对结果求平均.实验中使用的软件包括Spark-1.0.0,Shark-0.9.2与Hadoop-1.2.1.

3.1 写入性能测试

图7展示了传统RDBMS与基于Spark/Shark的OLAP系统在写入电力大数据时的性能.由结果可以看出,HDFS的数据写入性能比RDBMS高12倍.如果RDBMS的表中已建立索引,那么写入性能会进一步降低.其原因是:HDFS的写入操作是直接写入分布式文件系统,而在RDBMS中,为了更快地查询数据,数据一般使用B树组织,每写入一条记录,需要在B树中寻找记录要写入的页.如果表建立了索引,那么在数据写入过程中,还要维护索引的结构,会极大地影响数据写入的性能.

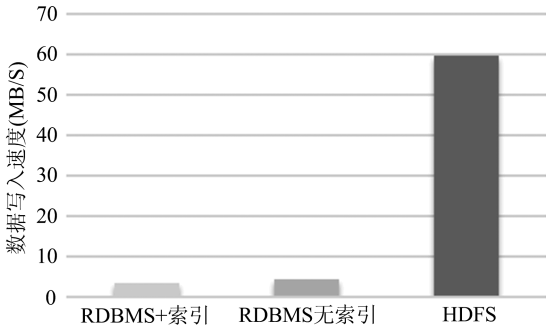

图7 RDBMS与OLAP系统数据写入性能结果

Fig.7 Write performance comparison of RDBMS and OLAP

3.2 OLAP查询性能测试

图8展示了原生Shark与使用了TrieIndex的Shark在处理电力大数据OLAP查询时的性能,纵轴使用对数形式.由于Shark可以调整处理每个查询的资源数,这里的资源数1表示使用1个CPU核,1 GB内存处理,其他数字以此类推.

从结果可以看出:①如果使用原生Shark,随着资源数的增加,查询耗时会递减,即增加1个查询所使用的资源数可以提高查询速度.在实际生产环境中,系统的总资源数是固定的,为某个查询分配较多资源则会影响其他查询的性能,并降低整个OLAP系统支持的的并发查询数.②对于同样的资源数,通过TrieIndex能将查询速度提高10倍以上.③当使

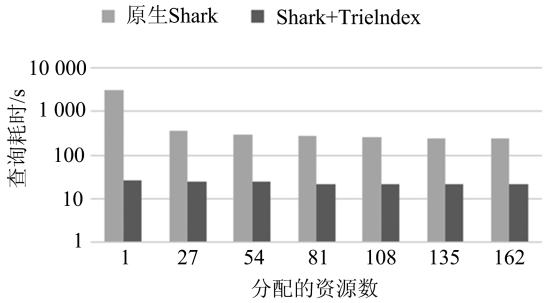

图8 原生Shark与Shark+TrieIndex查询性能结果

Fig.8 Query performance comparison of Shark and "Shark+TrieIndex"

用资源数为1时,即只使用1个CPU核,1 GB内存处理该查询时,原生Shark的耗时远远大于资源数较多的耗时,而基于TrieIndex索引方式的耗时与资源数较多的耗时基本相同.这是因为经过索引过滤,Shark只需读取很少的数据量,所以只需要资源极少.④该结果还说明,使用相同资源数时,由于使用TrieIndex索引占用的资源数很少,这样可以加大查询的并发度.

4 结论

本文设计了一种基于Spark/Shark的OLAP系统,用于处理基于电力大数据的查询.该系统具有高吞吐量的写入速度,并具有高效的查询处理能力.为进一步优化Shark的查询处理性能,本文设计了一种基于Trie树的索引结构,并提出了相应的数据重组织算法.实验表明,这些技术极大地提升了Shark在处理电力大数据OLAP查询时的性能.下一步工作会关注预计算技术与TrieIndex的结合以及Slice在数据文件中的放置算法问题.

参考文献(References)

[1] Apache Hadoop. Welcome to apache hadoop[EB/OL]. https://hadoop.apache.org/.

[2] Spark. Lightning-fast cluster computing[EB/OL]. https://spark.apache.org/.

[3] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets[C]// Proceedings of the 2nd USENIX Conference on Hot Ttopics in Cloud Computing. Boston, USA: USENIX, 2010: 10-14.

[4] Xin R S, Rosen J, Zaharia M, et al. Shark: SQL and rich analytics at scale[C]// Proceedings of the ACM SIGMOD International Conference on Management of

Data. New York, USA: ACM Press, 2013:13-24.

[5] Abouzeid A, Bajda-Pawlikowski K, Abadi D, et al. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads [J]. Proceedings of the VLDB Endowment, 2009, 2(1): 922-933.

[6] Jiang D W, Ooi B C, Shi L, et al. The performance of MapReduce: An in-depth study[J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 472-483.

[7] Dittrich J, Quiané-Ruiz J A, Jindal A, et al. Hadoop ++: Making a yellow elephant run like a cheetah (without it even noticing) [J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 515-529.

[8] Eltabakh M Y, Özcan F, Sismanis Y, et al. Eagle-eyed elephant: Split-oriented indexing in Hadoop[C]// Proceedings of the 16th International Conference on Extending Database Technology. Genoa, Italy: ACM Press, 2013: 89-100.

[9] Liu Y, Hu S L, Rabl T, et al. DGFIndex for smart grid: Enhancing hive with a cost-effective multidimensional range index[C]// 40th International Conference on VLDB. Hangzhou, China: ACM Press, 2014: 1496-1507.

[10] 宋振伟. 用电信息采集系统数据库的云存储设计[D]. 山东大学, 2014.

[11] 彭小圣,邓迪元,程时杰,等. 面向智能电网应用的电力大数据关键技术[J]. 中国电机工程学报. 2015, 35(3): 503-511.
Peng X S, Deng D Y, Cheng S J, et al. Key technologies of electric power big data and its application prospects in smart grid[J]. Proceedings of the CSEE, 2015, 35(3): 503-511.

[12] Apache Hive™[EB/OL]. http://hive.apache.org/.

[13] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[C]// Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation. ACM Press, 2004: 137-149.

[14] Apache Oozie. Apache Oozie workflow scheduler for Hadoop[EB/OL]. http://oozie.apache.org.