文章编号:1006-2467(2019)08-0978-05 DOI: 10.16183/j.cnki.jsjtu.2019.08.013

基于窗口函数和分布式集群的可视化学术搜索系统数据查询优化

罗希意， 霍晓阳， 傅洛伊
(上海交通大学 电子信息与电气工程学院，上海 200240)

摘 要：针对在密集分析型查询请求和海量数据的应用场景下传统关系型数据库 MySQL 性能不佳问题，提出了基于窗口函数(Window Function)的分析型查询优化方法，以分区(Partitioning)方法代替传统的分组(Group by)操作，并提出了基于分布式集群(SQL-on-Hadoop：SparkSQL)计算引擎的海量数据查询优化方法，采用内存列存储优化技术和 Spark 分布式集群计算以提高查询性能. 同时，以典型的分析型 SQL 查询实例验证了其有效性. 结果表明，所提出的查询优化方法能够显著提高查询性能. 与传统的关系型数据库 MySQL 相比，基于 SparkSQL 的查询优化方法的查询速度大幅提高，从而验证了其用于可视化学术搜索系统 AceMap 数据查询的正确性.

关键词：结构化查询语言；窗口函数；分布式计算；查询优化
中图分类号：TP 392 **文献标志码**：A

Query Optimization of Data Based on Window Function and Distributed Cluster in Visual Academic Search System

LUO Xiyi， *HUO Xiaoyang*， *FU Luoyi*
(School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: In order to address the issue of the poor performance of traditional MySQL database in application scenarios of densely analytical query requests and massive data processing, we proposed an approach based on window functions for analytical SQL query optimization. The approach replaces the fundamental grouping operation by the partitioning operation. In addition, we also designed distributed clusters based method for massive data query optimization, the method utilizes the in-memory columnar storage technology and Spark cluster's distributed computation to lift the query performance. Meanwhile, the validity of the proposed approaches has been verified by typical analytical SQL queries. Experimental results show that the proposed methods have improved the query performance significantly, as the query optimization based on SparkSQL has reduced the execution time by a wide margin compared to traditional relational database MySQL. These proved the effectiveness when the methods are applied in AceMap, a visual academic search system.

Key words: structured query language (SQL); window functions; distributed computation; query optimization

收稿日期:2017-10-19
基金项目:科技部重点研发计划(2018YFB1004703)，上海市科学技术委员会创新行动计划(17511105103，18510761200)
作者简介:罗希意(1987-)，男，重庆市人，硕士生，主要从事数据库和大数据处理研究.
通信作者:傅洛伊，女，特别副研究员，E-mail:fu-ly@cs.sjtu.edu.cn.

在全世界范围内，每年自然科学和社会科学领域都会产出数以百万计的知识文献，包括学术论文、科技报告和书籍等，并且呈爆发式速度增长. 面对学术大数据[1]时代浩如烟海的学术资源，有效地检索或查询是一项极为重要的工作，因此，美国计算机协会(ACM)和美国电气与电子工程师协会(IEEE)分别构建了论文的电子数据库，一些互联网科技企业如谷歌、微软和百度等分别推出了各自的学术搜索系统，以帮助科技工作者查询学术文献.

作为国内学术搜索研究领域的代表，可视化学术搜索系统 AceMap 以可视化的方式呈现了学术大数据中各学科学术领域论文之间的引用关系、论文作者之间的合作关系和师从关系、世界范围内各学术机构和个人发表论文的统计信息等. AceMap 系统已经收录约 1.2 亿篇学术论文的相关信息，包括论文、作者、学术机构、学术期刊和会议等学术实体及其关系，其数据主要来源于微软学术图谱[2]、ACM 和 IEEE 的公开数据库. 所用方法：利用实体关系模型对这些学术实体及其关系进行抽象分析，并通过关系型数据库 MySQL[3]来实现，生成的主要数据表为论文(Papers)、作者(Authors)、机构(Affiliations)、期刊(Journals)、会议(Conferences)、学术领域(Field of Study)、论文与作者及其机构的关系(Paper Author Affiliations)和论文引用关系(Paper References)等. 这些数据表包含上亿条的数据，依靠传统的 MySQL 数据库难以应对高并发的应用场景，而且需要满足 AceMap 搜索系统的可视化学术关系分析和实时统计功能的要求，仅凭借基于全文检索的搜索引擎技术还不够，需要基于实体关系模型的结构化查询语言(SQL)的查询技术，因而需要寻求合适的查询优化方法.

本文结合实际应用场景提出了基于窗口函数(Window Functions)[4]的查询优化方法和基于分布式集群(SQL-on-Hadoop：SparkSQL)计算引擎[5-6]的查询优化方法，通过提取一些典型、引起系统性能瓶颈的分析型 SQL 查询实例，以传统的关系型数据库 MySQL 上的查询执行时间为基准进行性能对比实验，从而验证所提出的查询优化方法的有效性和正确性.

1　基于窗口函数的查询优化方法

窗口函数查询优化是在实体关系模型中寻求解决方案，旨在单节点关系型数据库的框架下优化分析型 SQL 查询.

1.1　窗口函数

窗口函数又称为 OLAP[7]函数或分析函数，属于 SQL：2003 标准[8]的新增部分. 它通过提取数据集合中指定分组的行并用于聚合、排名或分析，所涉及的 SQL 查询主要面向各类聚合操作，常用的聚合函数包括数值求和函数(sum)、计数函数(count)、最小值函数(min)和最大值函数(max)等. 窗口函数使用分区来代替传统 SQL 查询的分组操作，使其能够在各个分组中进行多次聚合. 其输出包括基础数据行和聚合结果，而分组操作只返回聚合结果. 分区(Partitioning)、排序(Ordering)和分帧(Framing)是窗口函数的 3 个核心部分，它们构成了窗口函数在 SQL 查询中的语义和语法，三者之间的逻辑关系见图 1[4].

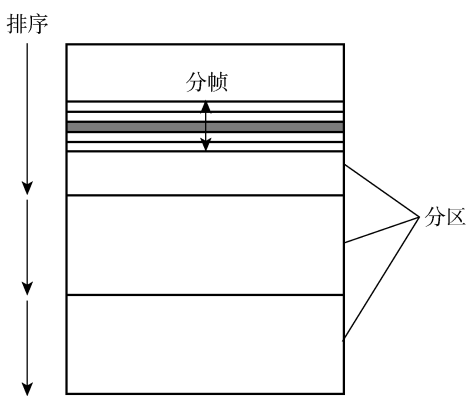

图 1　窗口函数的 3 个概念

Fig. 1　Three concepts of window function

(1) 分区. 分区是通过分区操作子句(Partition by)而实现的，是窗口函数的基础. 根据分区操作子句指定的数据表字段，将其中数值相同的数据行划分到同一个分区，以便于后续的聚合计算，但分区是执行逻辑分区，这与分组操作的物理分区不同. 例如，以 AuthorID 分区是将所有具有相同 AuthorID 的数据行划分到同一个分区.

(2) 排序. 排序是通过排序子句(Order by)而实现的，即将每一个分区中的数据行按照指定的数据表字段进行排序.

(3) 分帧. 分帧建立在分区的基础上，依赖于排序所确定的分区内数据行之间的顺序，具有行数(Rows)和数值范围(Range)两种限定模式，用于在数据分区内部划分局部的数据帧以供后续计算.

1.2　基于窗口函数的 SQL 查询优化

传统的 MySQL 数据库不支持窗口函数，因而

需要将存储学术大数据的数据库迁移至支持 SQL:2003 标准的数据库 PostgreSQL 9.4[9]上.

本文以 AceMap 系统中常见的分析型 SQL 查询(查询作者论文的 SCI 引用数)为例来验证基于窗口函数的查询优化方法.该实例查询涉及了一个大型的数据表 PaperAuthorAffiliations,以及存储论文、作者和机构的映射关系,包含 338 222 414 行数据记录.传统的 SQL 的查询程序为[4]

```
select count(*), sum(SCICitation) as sum
from PaperSciReferencesCount as tb1
inner join (select PaperID from PaperAuthor
    Affiliations where AuthorID=@AuthorID) as
    tb2
on tb1.PaperReferenceID=tb2.PaperID
group by AuthorID
```

采用基于窗口函数的查询优化方法的 SQL 查询程序为

```
select distinct count(*), sum(SCICitation)
    over(partition by AuthorID) as sum
from PaperSciReferencesCount as tb1
inner join PaperAuthorAffiliations as tb2
on tb1.PaperReferenceID=tb2.PaperID
where AuthorID=@AuthorID
```

1.3 性能对比实验

采用一台配置为两个中央处理器(因特尔至强系列,型号 E5-2630)、内存 128 GB 的服务器对 3 个学术系统中的典型 SQL 查询进行优化,并将查询执行时间作为性能评价指标.以原 SQL 查询在传统的 MySQL 数据库上的查询执行时间作为基准,对比采用基于窗口函数的查询优化方法的查询执行时间.表 1 列出了 3 个学术系统性能对比实验所涉及的 SQL 查询的具体内容,其查询执行时间的对比如图 2 所示.可以看出,采用基于窗口函数的查询优化方法能够在一定程度上提升查询性能,使其查询执行时间减少 18.6%(SQL-3).

表 1 窗口函数查询优化的 SQL 查询列表

Tab. 1 The SQL queries of window function query optimization

SQL 查询编号	SQL 查询描述
SQL-1	统计某一学者所有论文的 SCI 引用次数
SQL-2	统计学术合作次数,查找与某两名学者合作次数最多的学者
SQL-3	查找引用了某一篇论文的所有论文,并输出其中出现频次最多的关键词

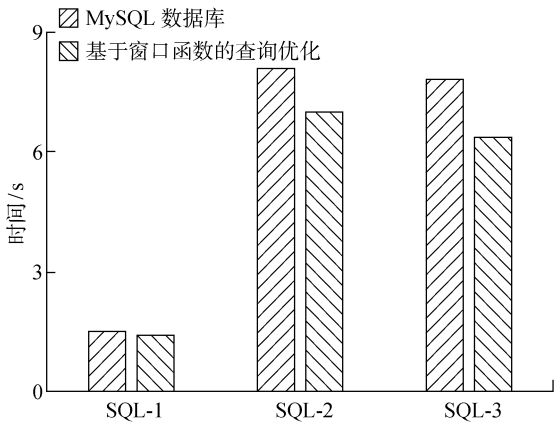

图 2 窗口函数的查询优化方法的结果

Fig. 2 Contrast results of window function query optimization

2 基于 SparkSQL 的查询优化方法

与基于窗口函数的查询优化方法的单节点计算模式不同,基于 SparkSQL 计算引擎的查询优化旨在分布式计算的框架下,借助于其多节点和多核的优势进行 SQL 查询,适用于大规模海量数据的应用场景.

2.1 Spark 与 SparkSQL

Spark[10]是类似于 Hadoop 的分布式计算系统 MapReduce 的计算引擎[11],通过分布式计算框架对大规模数据进行快速处理和计算.SparkSQL 计算引擎在 Spark 的生态系统中作为 SQL-on-Hadoop[11-12]系统存在,它是在分布式集群上进行 SQL 查询的技术平台.但是,SQL 查询不依赖于关系型引擎,它通过 SparkSQL 自带的编译器对 SQL 语言进行解析和编译并将其转化为 Spark 作业,再通过 Spark 计算引擎在集群上实现的.

2.2 SparkSQL 的文件系统和数据格式

SparkSQL 计算引擎需要分布式存储系统对其进行支撑.本文选择 HDFS(Hadoop Distributed File System)[11]作为 Spark 的文件系统.将数据由 MySQL 数据库迁移到 HDFS 上,即 SparkSQL 采用此分布式文件系统存取数据.

基于 HDFS 的文件系统中 Spark 支持多种数据格式,包括列式存储(Parquet)[10]和文本等.依据 Spark 计算引擎的官方技术文档[10],SparkSQL 在列式存储中的查询执行速度(文本格式查询执行速度的约 10 倍)最快,而且平均可以节省约 75%的存储空间,因此,本文选择列式存储作为 SparkSQL 的数据格式.

2.3　基于 SparkSQL 的查询优化

采用 SparkSQL 自带 Catalyst 优化器[10]能够实现对 SQL 查询的优化，并最终生成能够在 Spark 集群上执行的作业.

SparkSQL 采用内存列存储优化(In-Memory Columnar Storage)技术[13]，能够查询一些频繁出现的聚合，将其中包含的数据列存储到集群的同一个节点上，使其能够快速地被读入内存，从而提高查询速度.

参数优化是充分发挥 Spark 集群计算优势以高效执行作业的关键. 一个作业在 Spark 集群中的执行需要由位于 Spark 集群主节点中的 Driver 进程与若干个从节点中的 Executor 进程协同完成[10]，两者之间的关系如图 3 所示. 其中，Driver 负责集群资源的分配，Executor 负责具体任务的执行.

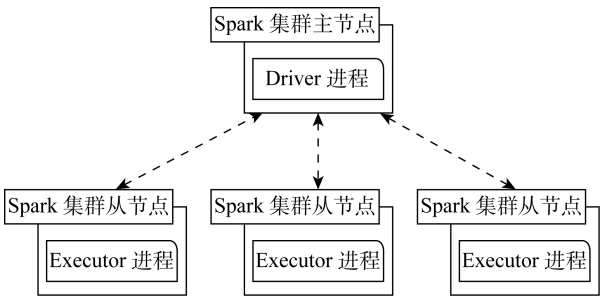

图 3　Driver 进程与 Executor 进程的关系

Fig. 3　Relationship between driver and executor processes

本文对以下参数[10]进行优化：

(1) 参数 SPARK_EXECUTOR_INSTANCES. 表示 Spark 集群能够同时启动的 Executor 实例个数的上限值，其对硬件资源的利用效率具有影响. 结合实验条件，本文最终选择其值为 20，经测试达到较优的查询性能.

(2) 参数 SPARK_EXECUTOR_CORES. 表示每个 Executor 能够使用的中央处理器(CPU)核的数量，它影响 Spark 执行任务时的并行度. 本文将其值设置为 10，相应的能够同时并行执行的任务数为 $20\times10=200$.

(3) 参数 SPARK_EXECUTOR_MEMORY. 表示分配给每一个 Executor 的内存数量，同一个 Executor 的所有 CPU 核共用. 经实际测试，将其值设置为 10 GB 时的查询性能较优.

2.4　性能对比实验

实验在一个含 4 个节点的 Spark 集群上进行，包括一个 Master 主节点，配置一个中央处理器(因特尔酷睿系列，型号 i5-4590，主频率 3.3 GHz)，内存 12 GB；3 个 Slave 从节点，配置均为两个中央处理器(因特尔至强系列，型号 E5-2630)，内存 128 GB.

本文对 3 个学术系统中涉及大规模数据的复杂 SQL 查询进行优化，并以查询执行时间作为性能评价指标. 以在传统的 MySQL 上的 SQL 查询执行时间为基准，对比通过 Spark 集群优化后的查询执行时间. 表 2 列出了实验所涉及的 SQL 查询的具体内容，其查询执行时间如图 4 所示. 可以看出，基于 SparkSQL 的查询优化方法能够大幅提升查询性能，使其查询执行时间降低 93.9%(SQL-2)，相当于查询速度加快 16 倍.

表 2　SparkSQL 查询优化的 SQL 查询列表

Tab. 2　The SQL queries of SparkSQL query optimization

SQL 查询编号	SQL 查询描述
SQL-4	统计分析学术领域之间的关系，查找与某一学术领域最为相关的领域
SQL-5	查找被某一学术领域引用次数最多的学者
SQL-6	统计某一学术领域引用论文的数量

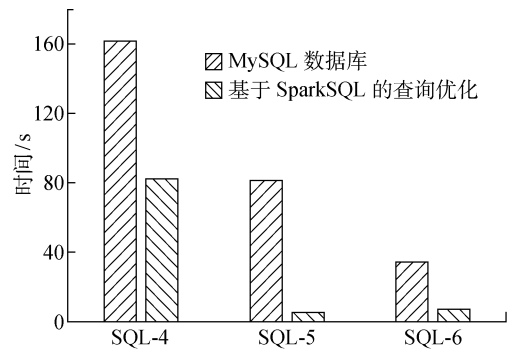

图 4　基于 SparkSQL 查询优化方法的查询执行时间

Fig. 4　Contrast experimental results of SparkSQL query optimization

3　结语

本文针对传统的关系型数据库 MySQL 的查询速度较低的问题，利用基于窗口函数的查询优化方法和基于 SparkSQL 的查询优化技术分别对各自适宜的应用场景进行优化. 结果表明：在 MySQL 的框架下，对于分析型 SQL 的查询，利用基于窗口函数的查询优化方法能够在一定程度上提高查询速度；在分布式集群计算的框架下，针对海量数据和更为复杂的查询操作，采用基于 SparkSQL 的查询优化方法能够大幅提高系统的查询性能.

参考文献：

[1] WU Z H, WU J, KHABSA M, *et al*. Towards

building a scholarly big data platform: Challenges, lessons and opportunities[C]// **Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries**. Piscataway, USA: IEEE Press, 2014: 117-126.

[2] SINHA A, SHEN Z H, SONG Y, *et al*. An overview of Microsoft academic service (MAS) and applications[C]// **Proceedings of the 24th International Conference on World Wide Web**. New York, USA: ACM Press, 2015: 243-246.

[3] SCHWARTZ B, ZAITSEV P, TKACHENKO V. High performance MySQL [M]. 3rd edition. Sebastopol, USA: O'Reilly Media, 2012: 1-31.

[4] LEIS V, KUNDHIKANJANA K, KEMPER A, *et al*. Efficient processing of window functions in analytical SQL queries[C]// **Proceedings of the VLDB Endowment**. Hawaii, USA: Morgan Press, 2015, 8(10): 1058-1069.

[5] ARMBRUST M, XIN R, LIAN C, *et al*. Spark SQL: Relational data processing in spark[C]// **Proceedings of the ACM SIGMOD International Conference on Management of Data**. New York, USA: ACM Press, 2015: 1383-1394.

[6] GUO C H, WU Z G, HE Z Y, *et al*. An adaptive data partitioning scheme for accelerating exploratory spark SQL queries[C]// **International Conference on Database Systems for Advanced Applications**. Suzhou, China: DASFAA Press, 2017: 114-128.

[7] CUZZOCREA A, BELLATRECHE L, SONG I. Data warehousing and OLAP over big data: Current challenges and future research directions[C]// **Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP**. New York, USA: ACM Press, 2013: 67-70.

[8] EISENBERG A, MELTON J, KULKARNI K, *et al*. SQL: 2003 has been published [EB/OL]. [2017-10-10]. https://dl.acm.org/citation.cfm?id=974142.

[9] RIGGS S, CIOLLI G, KROSING H, *et al*. PostgreSQL 9 Administration cookbook [M]. 2nd edition. Birmingham, UK: Packt Press, 2015: 1-8.

[10] AGARWAL S, ARMBRUST M, BRADLEY J, *et al*. Spark documentations-spark overview [EB/OL]. [2017-10-10]. http://spark.apache.org/committers.html.

[11] WHITE T. Hadoop: The definitive guide[M]. 4th edition. Sebastopol, USA: O'Reilly Media, 2015: 3-17.

[12] ABADI D, BABU S, ÖZCAN F, *et al*. Tutorial: SQL-on-hadoop systems [C]// **Proceedings of the VLDB Endowment**. Hawaii, USA: VLBD Press, 2015, 8(12): 2050-2051.

[13] PLATTNER H, ZEIER A, TINNEFELD C, *et al*. Available-to-promise on an in-memory column store: EP 2575093 [P]. 2011-10-10[2013-04-03].

(本文编辑:何雪)