基于分布式无共享架构的海量数据并行查询平台

秦东明[1] 喻 剑[1] 张 波[2] 赵 勤[1,2]
(同济大学嵌入式系统与服务计算教育部重点实验室 上海 200092)[1]
(上海师范大学信息与机电工程学院 上海 200234)[2]

摘 要 针对海量数据查询所面对的数据加载和并行查询控制等难题,提出了一种基于分布式无共享架构的海量数据并行查询平台。该平台利用分布式无共享架构为海量数据查询提供结构化与非结构化数据的统一处理,实现平台内数据的聚合计算。平台的核心技术如下:首先提供了多类型数据的跨平台存储与统一数据加载;然后给出了基于负载均衡的多节点数据查询任务流分配技术,生成全局查询执行策略;最后采用Hash和Range两种方式实现查询任务流的并发控制。根据测试验证,本技术在查询时间上相比于无并行方式节约了近40%。实验结果表明,该技术在海量数据查询的正确性、可靠性、并发性上具有较好的性能。

关键词 海量数据查询,无共享结构,并发查询,数据加载
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.04.007

Massive Data Parallel Query Platform Based on Distributed Shared-nothing Architecture

QIN Dong-ming[1] YU Jian[1] ZHANG Bo[2] ZHAO Qin[1,2]
(Key Laboratory of Embedded System and Service Computing of Ministry of Education, Tongji University, Shanghai 200092, China)[1]
(College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai 200234, China)[2]

Abstract In view of the challenges of data loading and parallel query controlling in massive data query systems, this paper proposed a massive parallel data query platform based on distributed shared-nothing architecture. The platform uses the distributed shared-nothing architecture to support the unified processing of structured and unstructured data in massive data query, and then achieves the aggregated calculation of data in the platform. The key technologies of the proposed platform are as follows. Firstly, the platform provides cross-platform storage and unified data loading of multiple types of data. Then, a multiple-node data query task flow distribution technology is proposed based on load balancing, and a global query execution strategy is generated. Finally, the platform uses Hash and Range methods to achieve parallel controlling for the query task flow. According to the performance verification of the proposed platform, the query time consumption of this platform is saved by 40% compared with nonparallel method. The experimental results show that this platform has good performance in the accuracy, reliability and concurrency of massive data query.

Keywords Massive data query, Shared-nothing architecture, Parallel query, Data loading

1 引言

进入大数据时代,人们面临着"海量数据查询"的难题。NoSQL技术的大量普及促使大数据查询技术快速发展,支撑大数据查询的系统也越来越多。许多现有的研究聚焦于如何通过集群数据系统中资源的融合与管理实现海量数据的高效查询[1-3]。

大数据系统大多是基于多节点集群架构的,其上的各节点(数据库)以分布式方式管理海量数据。对集群中各节点的管理方式主要包括:全透明共享(Shared Everything Architecture)、磁盘共享架构(Shared Disk Architecture)和无共享架构(Shared Nothing Architecture)等[3-9]。相比较而言,单一数据库(如SQL Server)所采用的全透明共享架构一般基于采用全局的透明共享CPU/MEMORY/IO等资源的策略[4],大多用于单一节点数据的处理,这种特点导致并行处理能力较低;磁盘共享架构中各个数据节点可以通过私有的资源[5]如

到稿日期:2018-03-17 返修日期:2018-06-23 本文受国家重点研发计划高性能计算专项(2016YFB0200300),国家自然科学基金(61572326,61702333),同济大学嵌入式系统与服务计算教育部重点实验室开放课题(ESSCKF 2016-01),上海市科委地方院校能力建设项目(17070502800)资助。

秦东明(1981-),男,博士生,工程师,主要研究方向为大数据处理、服务计算;喻 剑(1975-),男,博士,工程师,主要研究方向为数据挖掘、服务计算;张 波(1978-),男,博士,教授,主要研究方向为机器学习、社交网络分析,大数据分析;赵 勤(1982-),男,博士,讲师,CCF会员,主要研究方向为社交网络分析、数据挖掘、机器学习,E-mail:q_zhao@shnu.edu.cn(通信作者)。

内存等，保障数据处理能力，并实现数据共享，这也意味着该架构有一定的并行处理能力，但其依赖于存储器接口的性能，即存储器接口饱和后数据系统的整体性能将遇到瓶颈；分布式无共享架构中，各数据节点配置私有的资源管理机制，无需各节点间共享资源，各数据节点之间利用分布式通信协议实现协作通信，各节点之间无需串行分配资源，从而可以获得良好的并行处理和数据资源扩展能力[6-9]。目前，已有很多分布式无共享架构被广泛使用，如阿里的 OceanBase 架构、IBM 的 GPFS-SNC（General Parallel File System- Shared Nothing Cluster）架构[16-17]。这些系统主要针对各自具体的数据存储与查询等特殊需求所设计，能够较好地完成所对应的特定工作。然而，目前市场上尚缺少分布式无共享架构的通用数据解决方案，也就是说，无共享架构下仍面临着不同类型的海量数据加载[6,10]、数据流查询[1,4]、查询连接[11-13]以及并发控制[8-9]等个性化难题，现有主要方案均重点突出特殊问题的优化处理特性。同时，一些常用的分布式无共享架构，如 OceanBase，存在多个查询处理节点，这些查询节点在并行连接时无法实现协同工作，在处理大表数据时效率不高。因此，如何基于无共享架构实现高效率/高兼容性的并行数据查询，已经成为当前众多大数据系统的主要挑战。

另一方面，海量数据查询面临着效率提升的瓶颈问题。1）数据加载是高效率大数据查询的前提，必须解决海量数据在分布式环境中的存放和加载难题。传统的数据加载方式通常只考虑了系统的存储容量是否满足加载需求的问题，未能在加载时就采取有效的数据划分方法。因此需要研究数据加载技术，通过有效的数据划分策略，使得不同规模的数据都能够在分布式系统中尽可能均匀分布，从而提高统计和关联查询的效率。2）查询策略是分布式数据查询方案的关键问题。传统查询通常为两阶段的查询处理，即所有数据存储节点先将各自的查询结果写入查询服务器节点，查询服务器再对结果进行汇总。当中间结果较大时，查询服务器必然成为查询处理的瓶颈。因此，须研究更通用的查询处理，使得查询的执行能够直接在数据存储节点有效完成，而查询服务器不参与查询的执行，只完成对结果的搜集。3）为了使查询能够分布在各个节点中正确执行，需要研究有效的并行控制方法，从而使其既满足查询的高效率需求，又能保证返回查询结果的准确性。

本文针对上述问题，基于中科曙光 XData 大数据一体机系统提出一种通用海量数据并行查询平台，实现对海量的结构化数据和半（非）结构化数据的数据写入、存储组织以及查询处理。其底层数据节点采用无共享的结构，以满足数据读写并发度、计算并发度，以及良好的系统扩展性和可维护性。本文给出了平台中数据加载、查询处理和并行控制这 3 个关键技术模块，并进一步对系统进行了用例测试以表明该平台的有效性和可用性。

2　平台架构

基于分布式无共享架构的海量数据并行查询平台，是在传统海量结构化数据处理系统的基础上，增加了对 Hadoop 系统非结构化数据处理的支持，实现了结构化和非结构化数据的统一存储和处理[14-15]。其沿用了当前流行的无共享架构，这种并行架构可提供较高的聚合带宽（Port Trunks）和集成计算（Integrated Computation）能力，具有较强的系统扩展能力，同时也可以简化系统的设计[6-7]。无共享的架构方式与底层的 Hadoop/HBase/Hive 等系统结构相同，可以完全兼容[6]；在具体部署时，可将底层的数据节点安装成数据库（RDBMS 10g/11g，或者 postgresql 9.2），也可以将每个节点部署为 Hadoop（HDFS，HBase 或 Hive）的一个数据节点。通过平台的大数据处理中间件，该架构对外提供统一的数据访问服务，包括数据定义、数据加载和数据查询等各类服务。

图 1 是平台的系统层次结构说明，从下往上主要包括数据存储层、数据处理层、应用接口层与应用层 4 个层次，每一层只能与上面一层进行通信或者消息传递，不能跨层和其他层次进行通信。

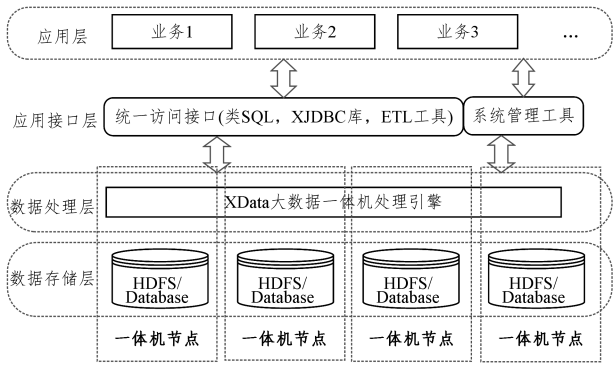

图 1　系统的整体架构

Fig. 1　System architecture

（1）数据存储层，是指特定的数据处理系统，主要包括数据库系统或者 Hadoop 系统。本平台支持的数据库系统包括 RDBMS 10g/11g，Postgresql 9.2，主要针对结构化数据的实时分析和查询处理。平台支持的 Hadoop 系统包括 Hadoop 生态系统的主要组件 HDFS，HBase，Hive 等，主要面向非结构化数据分析处理，以及结构化数据的离线分析处理。

（2）数据处理层，是指海量数据处理的服务程序，包括数据处理引擎和数据访问驱动。其中，数据处理引擎包括各类服务程序（数据定义服务、数据加载服务、数据查询服务等），以及内部的数据划分、存储组织和数据转换等模块。数据访问驱动主要是指对底层数据访问接口的封装，通过抽象的数据访问驱动，底层可以添加各种类型的存储系统。

（3）应用接口层，是指平台对外提供的接口，包括数据访问接口和系统管理工具两大类。数据访问接口包括统一的 XJDBC 接口和各服务模块的专用接口两种。XJDBC 接口的使用规范和方法类似于数据库的 JDBC 风格，可以满足各类数据处理业务的需求；各服务的专用接口可以直接与相应的服务程序进行通信，提高每一类服务的效率。系统管理工具，主要是针对数据处理层的各类服务和底层存储系统的安装部署、管理、配置、监控等业务的图形化管理功能的集成。

(4)应用层,是建立在平台之上的各种应用程序,通过平台提供的数据访问和管理入口,实现数据分析处理的业务。应用层不属于平台自身提供的功能,属于业务支撑范畴。

分布式查询平台通过层次划分明确各层的功能和接口,其中应用层通过接口层(包括 XJDBC 接口和管理接口)与数据处理层进行通信,不允许直接操纵数据存储层;而数据处理层通过抽象的数据访问接口,对数据存储层的数据进行访问和操纵。

3 关键技术

数据查询服务是面向海量数据的分布式查询平台的核心,其需要按照用户的类 SQL 的数据处理请求进行并行处理控制,得到最终的处理结果。该服务包括了数据加载、查询处理和并行控制三大模块。数据加载主要提供数据写入的功能,将一个或一组数据对象按照定义好的数据对象存储格式写入到底层存储系统中。数据加载必须提供的功能包括数据的批量传送、数据的划分和分发,以及将数据写入到底层存储系统等。非结构化数据库通常是实时或者定期地导入系统,因此必须提供批量加载的功能,并实现数据格式和数据存储模式的转换;查询处理引擎将查询任务分解成并行的查询任务流;并行处理引擎则包含一个并行任务解释器和任务执行控制器,负责计算的执行和同步、数据转换和传输,并行地执行查询任务流图,完成计算结果的采集。

3.1 数据加载

数据加载服务是将前端加载客户端提交的数据安全、准确、快速地存入存储平台中,以便于日后查询。在存放数据时,需要解决如下 3 个问题:

(1)数据跨平台(HDFS/HBase/Hive/Database)合理存储;

(2)为结构化、半结构化、非结构化数据提供统一、简单的数据存储方式;

(3)便于数据查询和检索。

因此,当数据传输至本平台后,服务端的数据加载服务在具体的功能上分为了两大模块:数据划分和数据分发。

(1)数据划分(Data Divide):根据用户加载的内容属性和各个存储平台的特性,按照一定的数据分布策略(如 HDFS/Hive/Database 的 hash,range,list 划分,以及 HBase 的列存储特性),将用户提交的数据写入到与底层数据存储平台相对应的本地缓存中。必须为每个数据存储平台/存储平台的每个分区/数据库存储节点维护一个独立的缓存,当缓存中的数据量达到一定值时,则触发相应的数据分发服务。

(2)数据分发(Data Dispatch):调用底层的数据访问接口,即文件写入接口或数据库访问接口,将对应数据缓存中的数据写入到各存储平台中。每一个数据缓存对应一个确定的数据存储平台,或数据存储平台的一个分区,或数据库存储节点。

数据传输、数据划分和数据分发过程中数据的组织传输过程如图 2 所示。

数据传输时将用户数据按列分批发送至查询服务器,查询服务器根据各列数据的属性和各存储单元的分区规则,将数据划分为不同存储单元下的不同分区值,并依次写入对应的各存储单元不同分区下的缓存中;当达到一定数据量时,进行数据分发,将各缓存中的数据并行存储到各存储单元中,每个存储单元按照数据的划分策略对应多个数据写入连接。

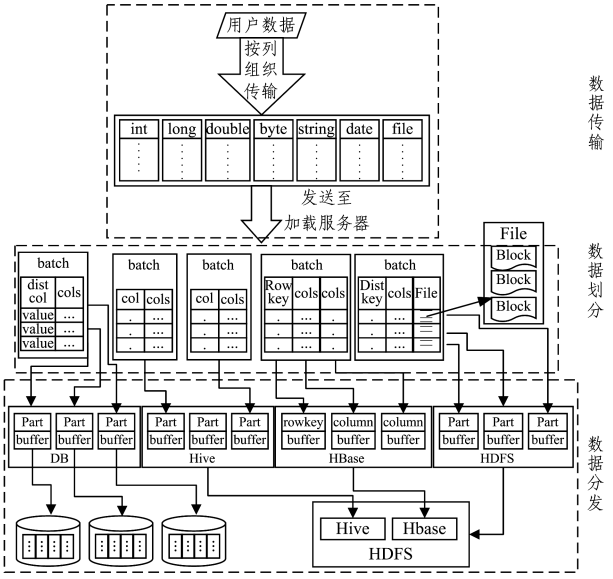

图 2 数据存储过程

Fig. 2 Process of Data Storing

3.2 查询处理

数据查询的流程如图 3(左侧)所示,执行步骤如下:

(1)数据查询服务启动后,在负载均衡器上注册。

(2)负载均衡器收到数据查询请求。

(3)均衡器判断作业的忙闲程度,分情况执行台下操作。

1)当作业数大于限定值(*MAX_REQUESTS*)时,负载均衡器会返回作业执行所需的大概时间,如 2 h,征求请求方是否继续执行的意见,请求方可以选择撤销此次请求,并返回状态(2);如果继续,则保留该请求,或者绑定一个相关服务给请求方。

2)当作业数小于或等于限定值(*MAX_REQUESTS*)时,绑定一个查询服务给请求方,进入状态(4);

(4)通过并行数据库查询接口将查询请求发送给数据查询服务器(查询服务器为数据库节点)。

(5)—(6)查询服务器生成并行查询执行计划,并启动并行查询的执行。

(7)并行查询执行框架在多个数据库节点执行查询任务流,包括并行执行和节点间的数据传输控制,查询服务器负责整体控制和任务同步。

(8)—(9)查询执行完毕后,将各节点的执行结果汇总到查询服务器节点。

(10)将结果返回给请求方。

如图 3 所示,在运行数据查询服务的整个过程中,为了保证数据查询的可靠性和正确性,中间件服务管理(Service Manager)模块需要及时得到数据库节点的当前状态,并将该状态以消息的方式发布给各个数据查询服务器。查询服务器可以根据最新的部署策略,生成全局的查询执行计划。

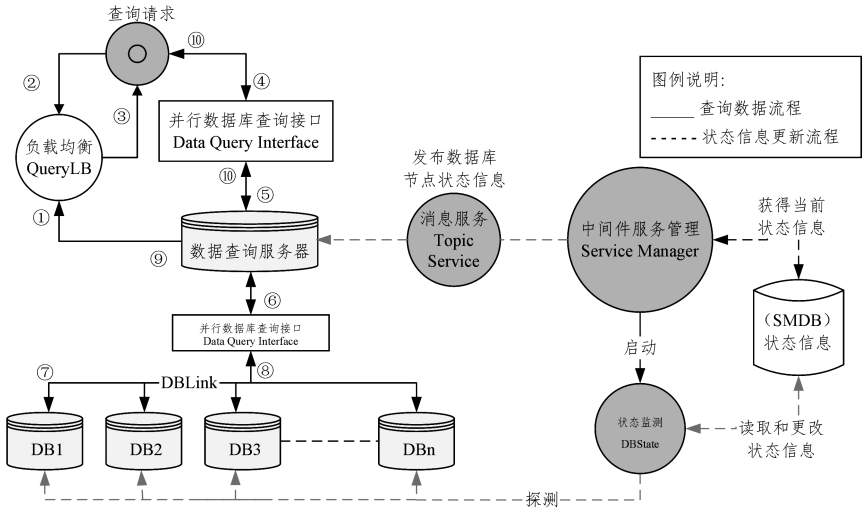

图 3　查询服务的流程

Fig. 3　Flow of query service

3.3　并行控制

在 3.2 节的查询流程中，第(7)步采用并行查询执行引擎运行生成的并行查询任务流。查询任务是一个由查询语句 Q_i、执行节点列表 SV、目的节点列表 DV 和数据分派方法 DD 组成的四元组，表示在原节点列表 SV 上执行该查询语句 Q_i，并将结果按照方法 DD 分派到目标节点 DV 上。而查询任务流图是指由查询任务组成的有向无环图，表示各任务执行的先后关系。如果采用 T_i 来表示每个查询任务，则查询任务流图可以表示为 $\{T_1,[S],T_2,[S],T_3,\cdots\}$，其中 $[S]$ 表示一次同步，可以根据语义按需加在两个任务之间。

查询任务可以用如下的四元组表示：{查询算子 OP；源节点列表 SV；目的节点列表 DV；数据分派方法 DD}。其含义是在 SV 节点组上并行地执行算子 OP，执行的结果以 DD 的方法分派到目的节点 DV 上。

查询任务流是由多个查询任务构成的一个有向无环图，描述的是各查询任务之间执行的先后顺序关系、同步方式以及数据传输关系。

数据节点之间的数据分派方法主要有 Hash 和 Range 两种方式。

(1)Hash 分派：默认的分派方法，将查询结果按照某一列的 Hash 值进行划分，其中 Hash 的 *key* 是指划分列的值(在实现时，均采用字符串类型)，*value* 是对应的目标节点的下标。在查询处理时，采用 MPI 程序进行控制和通信，每个数据库节点启动一个相应的 MPI 进程，因而 *value* 也同样指目标节点上的 Rank 值。

(2)Range 分派：Range 分派常用于对后续的数据进行排序的查询处理中。根据目标节点的数量进行划分，每个节点获得一个 Range 范围。如果采用 Range 划分，则在数据划分时，首先需要计算 Range 的切分点，采用估算法进行估算，计算头节点读取结果集的前 M(可配置)条记录，按照相应列的值进行统计划分，并将划分规则通过消息的方式发送给所有的 MPI 进程。对于 Range 划分，在后续的全局排序处理时，可以直接得到全局的排序结果，避免再次进行合并排序。

每一个用户 SQL 经过查询分析，均会生成一个查询任务流，交由并行查询引擎执行。并行查询引擎负责查询执行过程中的控制和节点间的数据交互。图 4 为一个并行查询任务流执行的控制流程示意图。

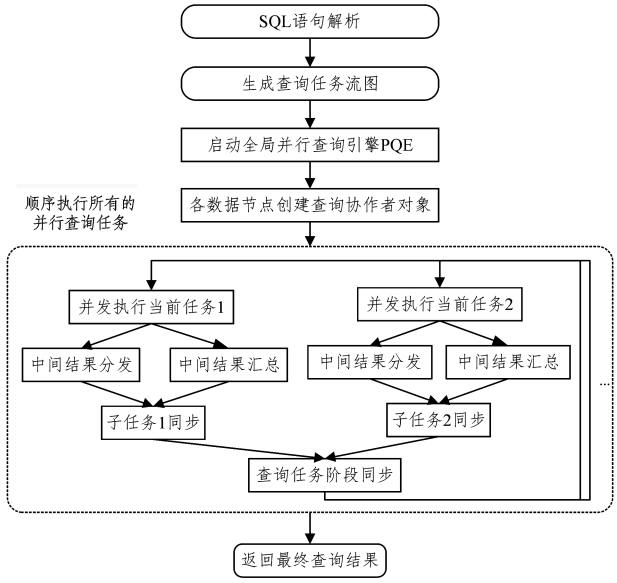

图 4　并行查询执行引擎的处理流程

Fig. 4　Processing flow of parallel query execution engine

在查询处理过程中，如果并行查询引擎出现故障，则查询服务器回收相应的资源，并重新创建新的并行查询引擎来执行该查询任务。资源回收和重新执行的过程对用户透明，如果重新执行依然出错，则返回相应的异常信息。

4　平台测试

将本文所提出的查询平台结构应用于中科曙光自主研发的 XData 系统。该系统是通用的海量数据处理平台，包含对海量的结构化和半(非)结构化数据的数据写入、存储组织以及查询处理。XData 底层数据节点采用本文所提出的无共享结构，实现 XData 底层多个数据存储节点的互联，进而采用本文所提出的数据处理技术实现并行处理计划，通过并行处理引擎完成数据查询和计算的统一。

为了检验本文所设计的数据加载和查询服务性能，利用 XData 中典型的 SQL 查询评测语句对海量数据分布式查询

平台的写入性能和并行查询效率进行详细的测试，验证并行查询处理的可用性。

4.1 测试环境

前端：运行客户端程序

数据库节点：运行 DBMS instance

配置情况：

12 * CPU AMD Opteron(tm) Processor 848 2.2GHz

MemTotal：16279148kB

Cache Size：1024kB

LSI409190 光纤主机适配卡

DISK：73GB(本地) LSI2882、JBOD、146GB * 28＝4TB

Suse Linux Enterprise Server 8，Kernel 2.4.19-SMP，Oracle 10.1.0.3.0

测试数据库实例的名称是 smdb。

主要表：TPCH 的 dss.dll 中表、外表和内表都只有简单表结构，没有索引和分区等。

测试方法：编写客户端测试程序，分别对外表和内表测试，并记录查询时间。每次测试完毕后清空内存，命令为：sync；echo 3＞/proc/sys/vm/drop_caches。

4.2 数据写入性能测试

1)写入文件测试：利用 tpch 的./dbgen-s 12 生成 12GB 的文件，在 3 个平台上的平均测试时间为 5min。

2)写入数据库测试：用 sql load 的方式写入数据库，把 12GB 的文件导入数据库，在 3 个平台上的平均测试时间为 70min。

具体测试结果如表 1 所列。由表中数据得出，字节数越多，插入数据库所用的时间就越长。写文件和写数据库的时间比为1∶14。

表 1 数据写入测试的结果

Table 1 Results of data writing test

字节数	表名	记录数/条	写入数据库时间/s	表列数
294048637	customer.tbl	1800000	62	8
9353294082	lineitem.tbl	71985077	3600	16
2224	nation.tbl	25	0.1	4
2101860804	orders.tbl	18000000	750	9
1448393354	partsupp.tbl	9600000	420	5
293115038	part.tbl	2400000	100	9
389	region.tbl	5	1	31
7032924	supplier.tbl	120000	8	7

4.3 并行查询性能测试

本文首先对单线程方式下内、外表插入和查询的平均耗时进行测试比较，结果如图 5 所示。从测试的结果来看，无论是大型数据表还是小型数据表，扫描一张整表时外表和内表的时间相近。

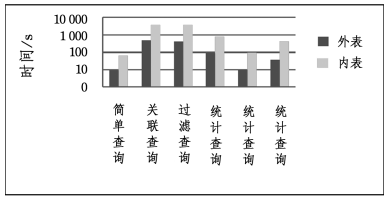

图 5 各类查询的内外表平均耗时比较

Fig. 5 Comparison of queries between inner table and outer table

然后测试并行查询的性能情况。首先使用 TPCH 生成的 SQL 语句测试单线程下无并行查询的总耗时，结果如图 6 所示，其中横坐标为测试的数据组别(下同)。从结果可以看出，内表所耗费的时间远多于外表所需要的时间。

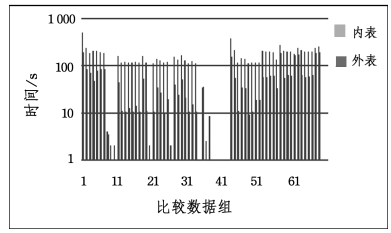

图 6 单线程方式下外表和内表插入与查询的总耗时比较

Fig. 6 Comparison of query time between inner table and outer table by using single thread mode

另外，从带有并行控制环境下的 SQL 语句测试结果(见图 7)来看，外表查询的时间消耗下降明显，而内表的时间消耗变化并不明显。

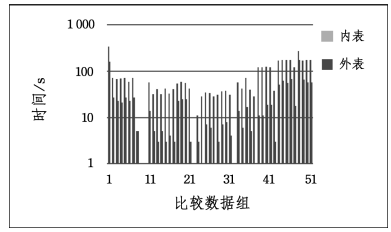

图 7 并行模式下外表和内表插入与查询的总耗时比较

Fig. 7 Comparison of query time between inner table and outer table by using parallel mode

通过总耗时的对比可以看出，本文平台所采用的并行查询处理模式能够有效处理多种类型的复杂查询，在查询时间上相比于无并行方式节约了近 40%。

为了测试外表在多表联合查询与单表查询情形下的耗时差异，针对相同数据量的外表，对将其存成多个文件和存成一个文件的情况进行比较。本测试采用带有 parallel 语句的查询，测试结果如图 8 所示。可以看出，多个文件的耗时比一个文件多。

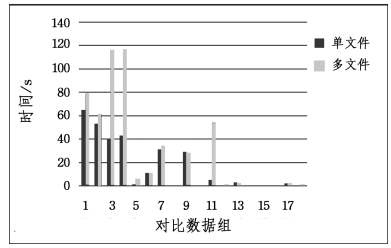

图 8 外表存为单文件和多文件的查询时间比较

Fig. 8 Comparison of query time by saving as single file and multiple files

从测试的总体情况来看，本文所提出的平台特征主要包括：1)通常外部表数据的写入速度是内部表的 10 倍以上；2)无论大表还是小表，扫描一张表时外表和内表的时间相差不多；3)条件复杂的情况下，一个查询语句如果要多次扫描一张表，那么外表就会比内表慢，但是时间远小于写入时间，因

此外表的总用时短。

结束语　本文阐述了基于无共享架构的海量数据分布式查询平台的系统架构、关键技术以及平台的相关测试。结果表明，基于分布式无共享架构的海量数据并行查询平台能够在保证数据写入性能的基础上，利用并行处理技术提供数据的查询性能，且能够使用多种复杂类型的查询，从而在实践中具有很强的通用性，可满足用户业务环境下的不同需求。本文所提出的平台目前被直接应用于中科曙光自主研发的 XData 一体化数据查询平台，未来将重点面向实时数据进行查询效率的优化提升。同时，由于不同系统之间的技术细节、软硬件配置、数据测试用例之间存在差异，文中没有进行横向的直接性能比较。

参 考 文 献

[1] WANG C K, MENG X F. Relational Query Techniques for Distributed Data Stream: A Survey[J]. Chinese Journal of Computers, 2016, 39(1): 80-96. (in Chinese)
王春凯，孟小峰. 分布式数据流关系查询技术研究[J]. 计算机学报，2016，39(1)：80-96.

[2] YANG B, YANG B, CHI Y, et al. Enabling Smart Transportation Systems: A Parallel Spatio-Temporal Database Approach [J]. IEEE Transactions on Computers, 2016, 65(5): 1377-1391.

[3] GAO Q, ZHANG F L, WANG R J, et al. Trajectory Big Data: A Review of Key Technologies in Data Processing[J]. Journal of Software, 2017, 28(4): 959-992. (in Chinese)
高强，张凤荔，王瑞锦，等. 轨迹大数据：数据处理关键技术研究综述[J]. 软件学报，2017，28(4)：959-992.

[4] TÖZÜN P. Scalable and dynamically balanced shared-everything OLTP with physiological partitioning[J]. Vldb Journal, 2013, 22(2): 151-175.

[5] OHN K, CHO H. Path conscious caching of B tree indexes in a shared disks cluster[J]. Journal of Parallel & Distributed Computing, 2007, 67(3): 286-301.

[6] JIN S D, FENG Y C. Parallel DBMS Architecture Supporting Shared-Nothing Computers[J]. Computer Research and Development, 1998, 35(6): 520-524. (in Chinese)
金树东，冯玉才. 支持无共享结构的并行 DBMS 软件结构[J]. 计算机研究与发展，1998，35(6)：520-524.

[7] CHUNG W, PARK S Y, BAE H Y. Efficient Parallel Spatial Join Processing Method in a Shared-Nothing Database Cluster System[J]. Lecture Notes in Computer Science, 2003, 3605(4): 81-87.

[8] DEWITT D J, NAUGHTON J F, SCHNEIDER D A. Parallel sorting on a shared-nothing architecture using probabilistic splitting[C]// International Conference on Parallel and Distributed Information Systems. IEEE, 1991: 280-291.

[9] CHAKRABORTY A, SINGH A. Parallelizing Windowed Stream Joins in a Shared-Nothing Cluster[J]. IEEE, 2013: 1-5.

[10] YANG J J, LIAO Z F, FENG C C. Survey on Big Data Storage Framework and Algorithm[J]. Journal of Computer Application, 2016, 26(9): 2465-2471. (in Chinese)
杨俊杰，廖卓凡，冯超超. 大数据存储架构和算法研究综述[J]. 计算机应用，2016，36(9)：2465-2471.

[11] CI X, MA Y Z, MENG X F. Method for Top-K Query on Big Data in Cloud[J]. Journal of Software, 2014, (4): 813-825. (in Chinese)
慈祥，马友忠，孟小峰. 一种云环境下的大数据 Top-K 查询方法[J]. 软件学报，2014，25(4)：813-825.

[12] ZHU R, WANG B, YANG X C, et al. Indexing Probabilistic Data for supporting Range Query over Big Data[J]. Chinese Journal of Computers, 2016, 39(10): 1929-1946. (in Chinese)
朱睿，王斌，杨晓春，等. 大数据环境下支持概率数据范围查询索引的研究[J]. 计算机学报，2016，39(10)：1929-1946.

[13] ZHAO Y R, WANG W P, MENG D, et al. Efficient Join Query Processing Algorithm CHMJ Based on Hadoop[J]. Journal of Software, 2012, 23(8): 2032-2041. (in Chinese)
赵彦荣，王伟平，孟丹，等. 基于 Hadoop 的高效连接查询处理算法 CHMJ[J]. 软件学报，2012，23(8)：2032-2041.

[14] REN Z, WAN J, SHI W, et al. Workload Analysis, Implications, and Optimization on a Production Hadoop Cluster: A Case Study on Taobao [J]. IEEE Transactions on Services Computing, 2014, 7(2): 307-321.

[15] ZELENKAUSKAITE A, SIMOES B. Big data through cross-platform interest-based interactivity[C]// International Conference on Big Data and Smart Computing. IEEE, 2014: 191-196 [J]. Journal of Software, 2014, 37(1): 189-206.

[16] YANG Z K. The Architecture of OceanBase Relational Database System [J]. Journal of East Normal University (Natrual Science), 2014(5): 141-148. (in Chinese)
阳振坤. OceanBase 关系数据库架构[J]. 华东师范大学学报（自然科学版），2014(5)：141-148.

[17] GUPTA K, JAIN R, KOLTSIDAS I, et al. GPFS-SNC: An enterprise storage framework for virtual-machine clouds[J]. IBM Journal of Research and Development, 2011, 55(6): 2: 1-2: 10.