基于列存储的 MapReduce 分布式 Hash 连接算法

张 滨[1] 乐嘉锦[2]

(浙江财经大学 杭州 310018)[1] (东华大学计算机科学与技术学院 上海 201620)[2]

摘 要 大数据具有规模大、深度大、宽度大、处理时间短、硬件系统普通化、软件系统开源化的特点。传统关系型数据库在对大数据进行操作时存在系统性能严重下降、计算效率提升有限以及可扩展性差等问题,因此引入 MapReduce 并行计算模型,提出一种大数据上基于列存储的 MapReduce 分布式 Hash 连接算法。首先,设计面向大数据的分布式计算模型,在设计的分片聚集并行连接的基础上,利用 Hash 连接以及动态探测方法优化了数据并行连接处理效率;然后,针对该算法开发了基于 Hadoop 的原型系统。通过实验证明,在大数据分析处理中,所提算法在执行时间和负载能力上都有很好的性能表现,也能提供良好的可扩展性。

关键词 大数据,列存储,Hash 连接,MapReduce,并行计算

中图法分类号 TP311 **文献标识码** A

Hash Join in MapReduce Distributed Environment Based on Column-store

ZHANG Bin[1] LE Jia-jin[2]

(Zhejiang University of Finance & Economics, Hangzhou 310018, China)[1]

(School of Computer Science and Technology, Donghua University, Shanghai 201620, China)[2]

Abstract The characters of big data are volume, variety, value, velocity, and common hardware and open source. Aiming at the system inefficiency and limited scalability of traditional relational database in big data analysis, this paper presented an algorithm of Hash joins in MapReduce distributed environment based on column-store by introducing MapReduce computing model. First of all, this paper proposed the design of large data-oriented distributed computing models. Then, it proposed the partition aggregation and the heuristic optimization strategy to realize the implementation of Hash join algorithm. Lastly, the experiments evaluated execution time and load capacity. The results show that the proposed method is effective and can provid good scalability in big data analysis.

Keywords Big data, Column-store, Hash join, MapReduce, Parallel computing

1 引言

随着计算机技术和互联网的急速发展,特别是随着 Web 2.0 的发展,互联网上的数据量快速增长,现有技术已无法对大数据进行处理。随着待处理数据增多,不仅不能将大数据存储在一台或有限数目的服务器内,而且不能由数目有限的计算机来进行处理。因此,如何实现资源和计算能力的分布式共享以及如何应对当前数据量高速增长的势头,是目前大数据研究的重点。

大数据的特点包括:处理时间短、硬件系统普通化、软件系统开源化。传统大数据处理技术主要有两大类:1)对称多处理机架构(SMP);2)大规模并行处理(MPP)架构。在数据量极具膨胀的大数据背景下,对数据处理的要求超出了单机乃至以小型机为代表的 SMP 架构处理能力,因此在大数据处理技术中计算分布和存储分布的 MPP 架构成为主流。MapReduce分布式并行计算框架[1]正是 MPP 架构的代表,具有高性能、灵活扩展、高容错和自动并行处理等特点。

另一方面,传统的关系运算(特别是数据的连接运算)在执行过程中会产生大量的中间结果,从而导致大量的系统开销,执行效率低下,这已经成为影响大数据分析处理的瓶颈。在传统数据库中数据连接方法包括排序合并连接和嵌套循环连接,这两种连接方法都存在明显缺陷。对于排序合并连接,如果两个表在施加了目标 SQL 中指定的谓词条件后得到的结果集很大且需要排序,那么排序合并连接的执行效率较差;而对于嵌套循环连接,如果驱动表所对应的驱动结果集的记录数很大,则将通过在被驱动表的连接列上添加索引来实现优化,此时使用嵌套循环连接的执行效率也较低。这两种算法在处理大数据时,无论是海量随机读还是海量排序,都是不能被接受的连接算法。为了解决这两种连接算法在上述情形下执行效率不高的问题,引入了 Hash 连接技术来提高查询效率。

在 Hash 连接技术中,存储在 Hash 桶中的记录并不是连接表的完整行记录,而是 SQL 查询中与连接表相关的连接列和查询列。如果在磁盘中存储的数据是按列存储的,则在 Hash 连接中顺序读入需要的连接列和查询列即可,而不需要读取所有的元组,可以有效避免传统 Hash 连接算法内存溢出的问题,从而提高了查询效率,这就是本文算法引入列存储技术的原因。另外,Hash 连接技术中经常出现数据偏斜

本文受浙江省哲学社会科学规划课题基金(17NDJC179YB)资助。

张 滨(1978—),男,博士,讲师,CCF 会员,主要研究方向为大数据,E-mail:zhangbin@zufe.edu.cn(通信作者)。

(Skew)的问题,即连接列的数据分布不均匀,导致产生的Hash桶非常集中,从而降低了匹配速度。如果数据列分布偏移严重,Hash连接算法反而比传统算法慢。为了解决数据偏斜问题,我们在基于列存储的MapReduce分布式环境中设计了协同定位策略,在数据加载过程中对数据进行预处理,将数据列尽可能地均匀分布在每个并行计算节点上,以一定磁盘空间为代价来保证Hash连接算法的优化效率。

本文提出一种基于列存储的MapReduce分布式Hash连接算法,该算法在MapReduce分布式并行处理环境中结合列存储技术并利用Hash连接实现优化查询执行计划。本文第2节介绍了列存储技术的发展背景以及MapReduce与数据库技术相结合的研究现状;第3节给出了关于本文算法的相关定义和解释,详细阐述了基于列存储的MapReduce分布式Hash连接算法的设计以及动态探测优化方法;第4节阐述了原型系统和并通过实验对算法进行有效性、可扩展性和正确性的分析验证;最后总结全文。

2 相关研究工作

2.1 列存储

列存储[2]的概念可以追溯到20世纪70年代,早在1976年加拿大统计局开发实现了列存储数据库管理系统,并在20世纪80年代广泛应用。这些系统对传统DBMS底层存储进行修改,对关系表进行垂直分解,然而在查询初始时将查询涉及的列组装成行,使用面向行的查询执行引擎进行查询执行,其效率有限。随着企业对分析型查询需求的快速增长,列存储的研究在近年取得了较大进展。Monet DB[3]和C-Store[4]是其中最有影响力的代表性成果。Monet DB由荷兰国家数学和计算机科学研究院(CWI)研究开发。C-Store由美国MIT、Yale、Brandeis大学、Brown大学以及UMass Boston大学等联合研究开发,在存储结构、查询优化、压缩等方面进行了技术创新。

2.2 面向大数据处理的MapReduce模型

2004年Google研究员通过对网页数据存储和并行分析处理进行研究后,提出了MapReduce计算模型,此后在ACM等多个期刊上转载。MapReduce计算模型为大数据分析处理问题提供了一个新的有效解决方法和途径。它将一个任务分成两阶段(Map和Reduce),并采用传统的并行排序归并(Sort-Merge)计算模式。2008年底,作为MapReduce开源实现Apache的Hadoop[5]项目,在发布后迅速得到广泛关注和使用。Hadoop的HDFS(Hadoop Distributed File System)是一种专门为MapReduce设计且支持多结构化数据的分布式文件系统,处理大数据时性能优越。

2.3 MapReduce与数据库技术的结合

在并行数据库与MapReduce模型相结合的理论研究方面,国外以耶鲁大学的数据库研究团队的HadoopDB[6]研究为代表,他们通过传统数据分析方法分析结构和非结构化数据的系统,在云环境中使用现有的SQL工具,组织分析大量的"多层结构"数据,消除数据孤岛。还有一些研究则是改变底层的存储模式,将行存储转换成能有效减少I/O的列存储。例如,Google的BigTable就采用了列簇的概念[8],Google的Dermel[9]也使用了列存储,针对互联网网页这种非结构化数据设计了嵌套数据模型,将原始网页数据转换成结构化数据,记录通过有限状态机实现组装,并设计了嵌套数据的类SQL查询语言。文献[10]设计了在MapReduce框架下的数据管理原型系统——Llama,该系统在底层使用一个创新的文件存储格式:CFiles。Llama利用浓密的查询计划树和排序PF组来解决MapReduce任务中的多表连接问题,避免了重排造成的数据复制并减少了MapReduce任务数。文献[11]利用列存储技术对MapReduce进行改进,阐述了列存储格式兼容Hadoop复制和调度约束机制,证明了列存储格式在实际工作负载条件下能够加快MapReduce任务的处理速度。

2.4 Hash连接技术

传统的并行连接技术带来了大规模随机读或者排序操作,该操作给予内存和缓存区空间很大压力。这类连接算法在处理大数据时都不能被接受。因此,对Hash Join连接技术的研究一直是数据库领域的研究热点。

文献[12]研究了基于Shared-Nothing结构的Grace Hash和Hybrid Hash等常用的并行连接算法,分析了影响查询响应时间的各种因素,在此基础上以多种硬件成分作为参数建立一种代价分析模型。文献[13]提出了一个基于分布式共享虚拟存储器技术(DSVM)的并行Hash连接算法,然后设计了一个并行连接算法的测试评价基准,评价和分析了该算法在均匀数据分布情况下的3种不同负载的性能比较和在偏斜数据分布情况下2种调度策略的算法性能。文献[14]仔细分析了典型的Hash连接算法的不同阶段,考虑了每一个阶段的不同实现,产生了一系列的Hash连接算法。这个简单的连接算法建立一个共享的Hash表,由于不需要对输入关系进行划分,仅需要很少的参数设置,因此它很容易对查询进行优化。文献[15]提出基于Radix-Join算法的Hash连接多线程执行框架,通过实例分析了影响多线程Radix-Join算法性能的因素。在此基础上,该算法优化了Hash连接多线程执行框架中的各种线程及其访问共享Cache的性能,优化了聚集连接时Hash连接算法的内存访问,分析了多线程聚集划分的加速比。此外,自Oracle 7.3版开始,Hash连接正式进入Oracle优化器,执行计划生成连接任务借助Hash算法,通过构建连带小规模的嵌套循环(Nest Loop)连接,利用内存空间实现了高速数据检索。

综上所述,目前尚未有在MapReduce分布式环境下发挥列存储优势和Hash连接优化的研究工作,对并行连接运算缺少深入分析,现有MapReduce计算框架采用传统的并行排序归并(Sort-Merge)连接算法的方式也很局限。此外,如果数据列分布不均匀,偏移(Skew)严重,那么Hash连接算法的效率将明显下降。针对上述问题,我们提出基于列存储的MapReduce环境下数据查询中的Hash连接算法,结合列存储技术和协同定位技术,使MapReduce分布式环境下的列数据在每个节点上的分布都是非常均匀的。此外,结合分片聚集和Hash连接负载探测方法进行性能优化,可以有效解决Hash连接执行时计算节点的负载均衡问题。该算法能最大程度地避免不必要的元组复制与数据传输产生的I/O操作,从而提高大数据处理效率。

3 基于列存储的Hash连接算法

为提高Hash连接在并行节点的计算效率,本节设计了MapReduce分片聚集方法,阐述了Hash连接以及动态探测优化算法的具体实现。

3.1　分片聚集方法

基于列存储的 Hash 连接算法所设计的分片聚集方法的示意图如图 1 所示。为了充分调用集群所有机器的计算资源来实现数据的高效并行连接[19]，在查询计划执行的 Map 阶段，本文提出分片聚集方法，其基本思想如下。

1）抽取：按照并行连接操作，在集群中的多个机器上分别执行完毕后得到子连接结果，将结果传递给分片聚集阶段。

2）分片聚集：该阶段对每个子连接结果进行聚集计算，利用分片方法来减少数据量，提高并行计算能力。而且，在多查询任务时分片聚集结果还可以重用。

3）分布：按照查询语句的分组条件，将前一阶段的结果重分配到各个分组中。这使得所有具有相同的查询字符串的结果被分配到同一个 Map 任务从而完成 GROUP BY 字句要求的对查询结果进行分组实现。

4）全聚集：每个分区即每个 Map 任务通过合并计算具有相同的查询字符串的查询结果，得到最终聚集结果，例如得到 count(*)结果。

5）过滤：过滤掉 HAVING 字句中的组条件。例如：count(*)＞50，计数小于 50 的将不会传入 Reduce 阶段。

6）排序：调用 Hadoop 的排序算法（这里调用 TeraSort 算法）对剩余的结果按照 ORDER BY 字句的要求分别并行排序。

7）合并：每个 Reducer 进行合并操作，将所有分区排序结果合并在一起，输出最终结果。

8）输出：输出 MCF 文件。

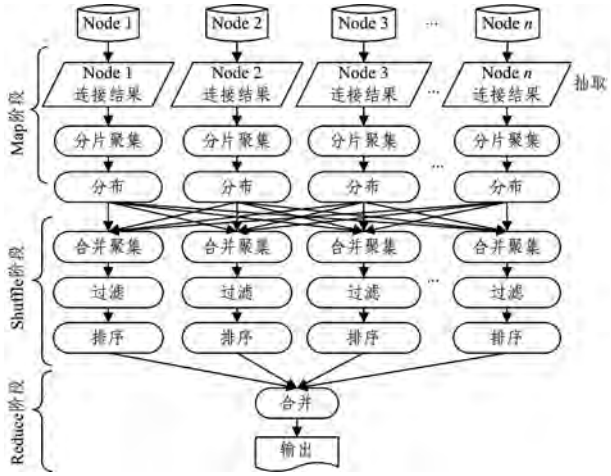

图 1　分片聚集查询计划图

3.2　Hash 连接算法

在相关研究工作中提到的归并排序（Merge Sort）带来的块读相对较少，但是付出的 CPU 成本和执行时间也是不可忽视的。将数据集合排序映射到内存中，消耗的每个节点的 CPU、内存和网络资源较多，排序阶段更是如此。针对上述问题，提出了 Hash 连接优化算法，该算法分为两个阶段：建立和连接，分别如算法 1 和算法 2 所示。

算法 1　Hash Building Phase

输入：objs：spatial objects of dataset A

输出：R：Result pairs

Data：PA：array of partitions of nodes/objects

nodes：array of nodes

1. R＝∅；
2.　Group the objects of A into p partitions PA；
3.　T＝Construct the hierarchical partitioning tree on PA；
4. Assign the objects of B to T；
5. foreach in ∈ innernodes of T do
6.　foreach leaf ∈ descendent leaf nodes of in do
7.　　R←R∪join(in. entities，leaf. entities)；
8. Return R；

算法 2　Hash Join Phase

输入：inner：an inner node

输出：R：a subset of result pairs

1. Uniformly divide the space to cells of same size；
2. foreach obj ∈ inner do
3. C＝cells that obj overlaps with；
4. foreach cell ∈ C do
5. cell. objects ← cell. objects ∪ obj；
6.　foreach L ∈ descendant leaf nodes of inner do
7. foreach obja ∈ L do
8. C＝cells that obja overlaps with；
9. foreach cell ∈ C do
10. foreach objb ∈ cell. objects do
11. if obja overlaps objb then
12. R ← R ∪ (obja，objb)；
13. Return R；

算法的基本思想如下：

Step1　每个节点的 MapReduce 的 Map 任务选择其中一个“小表”作为 Hash 连接驱动表（一般选择维表，其数据量相对较小），建立 Hash 表，把参与连接操作的连接属性作为哈希键，读取在 MCF 文件系统的驱动表中的连接属性列并存储到 MapReduce 调度系统的 datanode 内存中，由于每个节点的 MCF 存储的数据列大小远小于系统内存，因此不会出现传统关系数据库中 Hash 连接内存溢出的情况。然后，对连接列字段的所有数据值进行 Hash 函数操作。

Step2　将经过 Hash 处理的驱动表连接列与数据一起存放到在该内存中开辟的一块专门存放此类数据的空间，命名为 Hash_area。然后，依据不同的 Hash 函数值对驱动表进行划分桶操作，每个桶中包括所有相同 Hash 函数值的驱动表数据。

Step3　算法将每个节点上进行 Hash 连接的“大表”（一般是事实表，作为被驱动表）将其要做连接的数据列从 MCF 中依次分批读取，并且对连接属性列做 Hash 运算，确定应该到哪个桶中去查找，从而利用简单的 Hash 检索算法定位到适当的桶上；

Step4　在定位到的桶中，进行小规模的精确匹配，得到符合条件的行号。

Step5　对于每个节点符合条件的行号，由 MapReduce 的 Reduce 任务做合并，在 MCF 文件系统中读取 SQL 语句中涉及的查询列，最后输出查询结果。

Step1 和 Step2 是 Hash 连接的建立阶段；Step3 和 Step4 是 Hash 连接的连接阶段。

驱动表的数据列在每个计算节点上将被复制多份；而被驱动表的数据列已经被水平划分，分别储存在各个节点上。因此，Map 任务直接在每个节点做 Hash 连接，最后由 Reduce

做归并即可，不需要同传统算法一样将数据复制到远程处理器上，从而大大减少了计算过程中数据复制的时间开销，减轻了网络负载。由此可见，因为算法的搜索范围明显缩小，所以进行匹配的成功率精确度高。同时，匹配操作在内存中进行，其速度较合并排序快得多，从而实现了优化。

3.3 负载数据偏斜的动态探测

除了 3.3 节利用协同定位优化策略解决的连接属性值的不均匀分布导致内在数据偏斜的问题，还有一种情况就是在每个并行计算节点间隐含着的连接中的计算负载不平衡会导致负载数据偏斜。下面通过 MapReduce 分布式环境下的动态探测方法将探测器建立在 MapReduce 的 Jobtracker 守护进程上，实时修正节点的计算负载，调度新的 Map 任务，平衡每个节点的负载，从而解决负载数据偏斜问题，优化 Hash 连接的动态性能。

在 MapReduce 分布式环境下设计负载数据偏斜的动态探测器。作为一个监视任务进程，其示意图如图 2 所示。

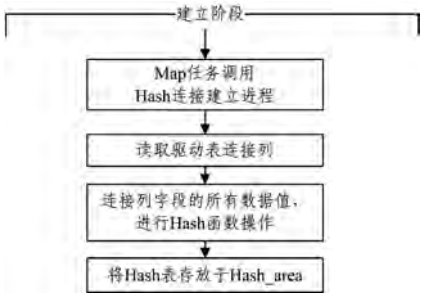

图 2 Hash 连接负载数据偏斜的动态探测建立阶段示意图

步骤定义如算法 3 所示。

算法 3 Hash Join Probe of load

输入：resource of nodes

输出：node of unload

1. PA＝partition objs into partitions of size fo；
2. while |PA|＞1 do
3. foreach partition p ∈ PA do
4. calculate MBR of p；
   make new node n with MBR
   set objects inside p as children of n；
   insert n into nodes；
5. PA＝partition nodes into partitions of size fo；
6. Return PA.

其算法步骤如下：

Step1 在建立阶段，将每个桶映射到一个节点上，并将其记录在负载数据偏斜的动态探测器中。

Step2 如图 3 所示，在连接阶段，探测器收集每个节点的负载数据，一旦发现不平衡情况，将该节点所映射的桶重新分配给新节点，新节点数目由负载情况确定。进行替代后原节点资源被回收，以便再次分配。

Step3 当每个节点的 Hash 连接完成后，动态探测操作节点和原非替代节点的数据，并将其一起交由 Reduce 阶段做合并，最后输出查询结果。

与传统 Hash 连接算法相比，该算法有效利用每个节点的计算资源，最大程度地避免不必要的元组复制与网络数据传输产生的 I/O 操作，从而提高大数据处理效率。

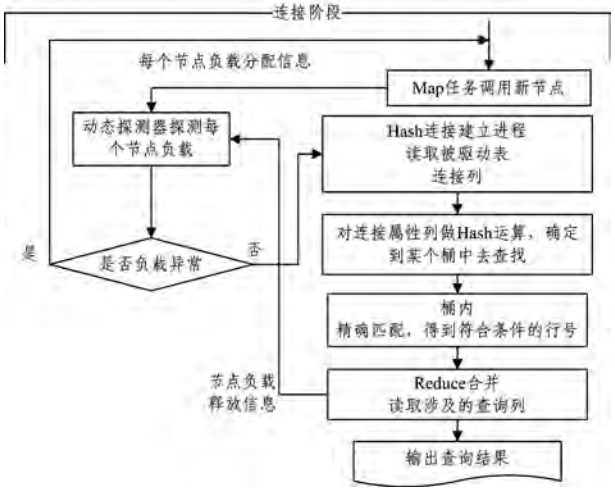

图 3 Hash 连接负载数据偏斜的动态探测连接阶段示意图

4 实验验证

实验采用课题组开发的原型系统 HCMS(Hadoop Column-store Management System)。HCMS 在 Hadoop 开源 MapReduce 计算平台上实现了基于列存储的 MCF 文件存储，在计划生成器中使用提出的基于列存储的 MapReduce 分布式 Hash 连接算法，采用国际通用的 SSB 测试数据集[16]对算法进行测试，从而验证了其高效性和可扩展性。

4.1 实验环境

本实验环境与原有单机环境不同，对计算机数量有更高要求。实验系统运行在课题组实验室选取的 50 台普通计算机组成的测试集群上，每个节点具有 4 核 CPU，4 GB 主存，1 块 500 GB SATA 硬盘，每台机器的操作系统都是 Redhat Linux 6.1。网络环境为 1Gb 以太网交换机组成的局域网。本实验使用的软件环境如表 1 所列。此外，DBMS3.0 为课题组前期在列存储数据库方面的研究成果。实验规划每个Datanode分配 6 个 map 任务和 2 个 reducer 任务。设置 HDFS 数据块的大小为 256MB，MapReduce 查询执行器使用全局内存大小为 1GB。

测试节点从 10 个节点开始增加到 50 个，记录每个测试数据集合和测试查询语句的执行时间和日志。每次测试完成后需要对集群 HDFS 重新格式化。

表 1 实验软件环境

编号	软件	软件版本
1	操作系统	红帽企业 Redhat Linux. 6.1
2	开发环境	Java jdk 1.6.31
3	MapReduce 软件	Hadoop-1.0.2
4	对比数据库软件 1	Hive-0.9.0

4.2 测试数据集

实验采用国际通用的星型模式基准 SSB 中定义的测试数据集进行大数据处理的实验验证，生成的星型模式下的真实数据集与基于 MapReduce 的大数据并行处理原型系统考虑的大数据这个测试目标相吻合。

实验采用 SSB 提供的数据产生器 DBgen 生成 SSB 的数据集实例。每个实例数据集的大小用增量因子控制，记为 SF，选用 $SF=1$，数据集大小为 1GB，初始 lineorder 表数据量为 6000000 行，如表 2 所列。实验逐步增加 SF，生成数据集大小为 10GB，100GB 和 1TB，分别记录每次测试结果。

表 2　实验测试数据

编号	表名称	元组数据量
1	customer	30000 * SF
2	date	2556 * SF
3	supplier	2000 * SF
4	part	2000 * [1+log2SF]

4.3　实验结果与分析

实验 1　各个测试语句性能对比

选取 SSB 的连接语句测试 Q1.1、简单聚集任务测试Q2.1 以及复杂聚集任务测试 Q3.1 和 Q4.1 作为基础测试语句，在全部节点都启用的条件下，通过对 100GB 数据分别测试 10 次来计算平均值，其结果如图 4 所示。

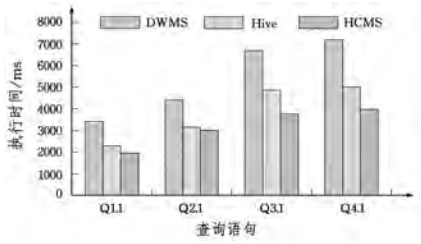

图 4　各个测试语句的性能对比图

由于采用了分片聚集算法，在聚集任务 Q3.1 和 Q4.1 (特别是复杂聚集任务)上，本算法的优化效果明显。

实验 2　当数据量变化时的性能对比

选取 SSB 的 Q2.2 作为基础测试语句，在全部节点都启用的条件下，通过对 10GB，100 GB 和 1 TB 数据分别测试 10 次来计算平均值，其性能对比结果如图 5 所示，测试系统在大数据条件下的运行负载能力。从图 5 中可看出，当数据量增加时，DWMS 的执行时间增长明显，而 Hive 和 HCMS 的执行时间则较平缓，充分验证了 MapReduce 并行执行的优越性。当负载增大到 1 TB 时，HCMS 的性能比 Hive 提高了 26.2%，由此充分验证了同时使用 MCF 列存储结构和分片聚集能进一步提升 Hash 连接的有效性，使得查询效率更高。

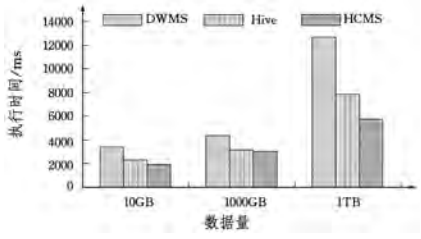

图 5　当数据量变化时性能的对比图

实验 3　当集群数量变化的性能对比

选取 SSB 的 Q2.2 作为基础测试语句，选择 100 GB 数据和 1 TB 数据，每次实验将集群 Datanode 节点数目从 10，20，30，40 增加到 50 个，完成重新格式化后，分别测试 10 次，计算平均值，其性能对比图如图 6 所示。列存储数据仓库管理系统 DWMS 不是并行分布式系统，在本次测试实验中未使用。

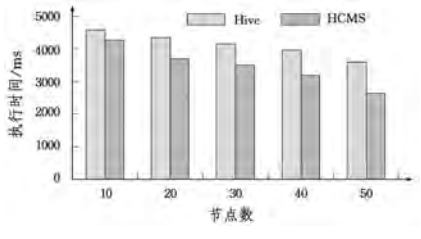

图 6　当集群数量变化时的性能对比图

由图 6 所示，随着运算节点数目的增加，HCMS 的执行时间比 Hive 更短，分别从 10 个节点的相差 15.8%，减少到 50 个节点的 26.3%，可见其优化效果更加明显，这也充分验证了 Hash 连接对性能的优化是非常明显的，另一方面也证实了算法的可扩展性。

结束语　基于列存储的 MapReduce 并行连接算法分析了在 MapReduce 并行环境下列存储连接与在单机环境下列存储的区别，重点抓住面向大数据的分布式计算模型“大而化小，分而治之”的设计思路，为大数据查询分析处理提供了有效的解决方案。算法在面向大数据的分布式计算模型的基础上，利用协同定位实现存储均匀分布优化，在分片聚集和 Hash 连接优化两个方面完善了算法的设计。实验证明，该算法有效地减少了 MapReduce 过程中的中间数据、网络带宽占用量和不必要的 I/O 开销。此外，由于利用了模型的可扩展性特点，该算法无论在执行时间还是负载能力上都有很好的性能表现，显著提高了大数据运算的效率。我们未来将对列存储的 MapReduce 连接索引技术进行进一步研究，对适用于列存储的 MapReduce 的各种索引进行分析研究，使 MapReduce 大数据查询性能得到进一步的优化。

参 考 文 献

[1] DEAN J，GHEMAWAT S. MapReduce：Simplified Data Processing on Large Clusters. [C]// 6th OSDI. San Francisco：USENIX Association，2004：137-150.

[2] ABADI D J，MADDEN S R，HACHEM N. Column-Stores vs. Row-Stores：How Different Are They Really? [C]// The 2008 ACM SIGMOD Int Conf. Vancouver，BC，Canada：ACM，2008：967-980.

[3] STONEBRAKER M，ABADI D J，BATKIN A，et al. C-Store：A column-oriented DBMS. [C]// VLDB Conference. Trondheim，Norway：VLDB Endowment，2005：553-564.

[4] BONCZ P，ZUKOWSKI M，NES N. MonetDB/X100：Hyper-Pipelining Query Execution. [C]// The Biennial Conf on Innovative Data Systems Research (CIDR). Asilomar，CA，USA：ACM，2005：225-237.

[5] BLANAS S，PATEL J M，ERCEGOVAC V，et al. A comparison of join algorithms for log processing in MapReduce[C]// The ACM SIGMOD International Conference on Management of Data. Indianapolis，Indiana，USA：ACM，2010：975-986.

[6] ABOUZEID A，BAJDA-PAWLIKOWSKI K，ABADI D J，et al. HadoopDB：An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads[C]// VLDB Conference. Lyon，France，VLDB Endowment，2009：922-933.

[7] BAJDA-PAWLIKOWSKI K，ABADI D J，SILBERSCHATZ A，et al. Efficient Processing of Data Warehousing Queries[C]// The ACM SIGMOD International Conference on Management of Data. Athens，Greece，ACM，2011：1165-1176.

[8] CHANG F，DEAN J，GHEMAWAT S，et al. Robert Gruber：Bigtable：A Distributed Storage System for Structured Data[C]// OSDI. 2006：205-218.

[9] MELNIK S，GUBAREV A，LONG J J，et al. Dremel：Interactive Analysis of Web-Scale Datasets[C]// VLDB Conference. Singapore，VLDB Endowment，2010：330-339.

(下转第 505 页)

依赖于专家的个人知识水平。这使得通过人工的方式建立装备-标准知识图谱在短时间内无法完成。因此，通过知识图谱过程建模厘清各实体单元之间的逻辑联系，在此基础上，借助高性能计算机自动、准确地将标准文档、标准条目、标准化要素、组件/模块、装备等实体进行关联，从而构建装备-标准知识图谱的数学模型是装备标准化建设的关键，后续将在这方面开展深入的建模研究。

参 考 文 献

[1] LIN Y，LIU Z，SUN M，et al. Learning entity and relation embeddings for knowledge graph completion[C]// Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015：2181-2187.

[2] JI G，LIU K，HE S，et al. Knowledge graph Completion with adaptive sparse transfer matrix[C]// Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. 2016：985-991.

[3] YANG S，ZOU L，WANG Z，et al. Efficiently answering technical questions-A knowledge graph approach[C]// Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. 2017：3111-3118.

[4] HEIKO P. Knowledge graph refinement：A survey of approaches and evaluation methods[J]. Semantic Web，2017，8(3)：489-508.

[5] NIKOLAEV F，KOTOV A，ZHILTSOV N. Parameterized fielded term dependence models for Ad-hoc entity retrieval from knowledge graph[C]// Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. Pisa，Italy，2016：435-444.

[6] BRACHMAN R J. What IS-A is and isn't：An analysis of taxonomic links in semantic networks [J]. Computer，1983，10：30-36.

[7] STEINER T，VERBORGH R，TRONCY R，et al. Adding realtime coverage to the Google knowledge graph[C]// Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914. CEUR-WS. org，2012：65-68.

[8] WANG Z，WANG Z，LI J，et al. Knowledge extraction from Chinese wiki encyclopedias[J]. Journal of Zhejiang University Science C，2012，13(4)：268-280.

[9] ZENG Y，WANG H，HAO H，et al. Statistical and structural analysis of web-based collaborative knowledge bases generated from Wiki Encyclopedia[C]// Proceedings of the 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01. IEEE Computer Society，2012：553-557.

[10] HUANG Z，CHUANG W，ONG T H，et al. A graph-based recommender system for digital library[C]// Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries. ACM，2002：65-73.

[11] SIEK J G，LEE L Q，LUMSDAINE A. The boost graph library：user guide and reference manual，portable documents[M]. Pearson Education，2001.

[12] DAI X，LI J，LIU T，et al. HRGRN：A graph search-empowered integrative database of Arabidopsis signaling transduction，metabolism and gene regulation networks[J]. Plant and Cell Physiology，2016，57(1)：e12.

[13] 刘峤，李杨，杨段宏，等. 知识图谱构建技术综述[J]. 计算机研究与发展，2016，53(3)：582-600.

[14] 徐增林，盛泳潘，贺丽荣，等. 知识图谱技术综述[J]. 电子科技大学学报，2016，45(4)：589-606.

[15] 耿霞，张继军，李蔚妍. 知识图谱构建技术综述[J]. 计算机科学，2014，41(7)：148-152.

[16] 刘昶，史海波，于海斌. 基于多 Agent 制造过程的建模方法[J]. 控制工程，2005，12(6)：515-519.

[17] 高飞，高阜乡，王钰，等. 基于实体的指挥控制过程仿真建模[J]. 指挥控制与仿真，2012，34(3)：116-120.

[18] 张树玲，孙波，田艳琴. 过程建模技术研究综述[C]// 2010 Third International Conference on Education Technology and Training. Wuhan，Hubei，2010：542-547.

[19] PLAIA A，CARRIE A. Application and assessment of IDEF3 - process flow description capture method[J]. International Journal of Operations & Production Management，1995，15(1)：63-73.

[20] 肖磊，金光，周忠宝，等. 基于 Petri 网的维修过程建模仿真方法研究[J]. 军械工程学院学报，2008，20(5)：6-11.

[21] SCHEER A W. Architecture of integrated information system-foundations of enterprise modelling[M]. Berlin：Springer-Verlag，1992.

（上接第 475 页）

[10] LIN Y T，AGRAWAL D，CHEN C，et al. Llama：Leveraging Columnar Storage for Scalable Join Processing in the MapReduce Framework[C]// The ACM SIGMOD International Conference on Management of Data. Athens，Greece：ACM，2011：961-972.

[11] FLORATOU A，PATEL J M，SHEKITA E J，et al. Column-Oriented Storage Techniques for MapReduce[J]. PVLDB，2011，4(7)：419-429.

[12] THUSOO A，SARMA J S，JAIN N，et al. Raghotham Murthy：Hive-A Warehousing Solution Over a Map-Reduce Framework. [C]// VLDB Conference. Lyon，France，VLDB Endowment，2009：1626-1629.

[13] HE Y Q，LEE R B，HUAI Y，et al. RCFile：A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems[C]// IEEE International Conference on Data Engineering. Hannover，Germany，2011：1199-1208.

[14] HSIAO H，CHEN M S，YU P S. Parallel execution of hash joins in parallel databases[J]. IEEE Trans. on Parallel and Distributed Systems，1997，8(8)：872-883.

[15] BONCZ P，MANEGOLD S，KERSTEN M L. Database architecture optimized for the new bottleneck：memory access[C]// The 25th Int'l Conf. on Very Large Data Bases. ACM Press，1999：231-246.

[16] O'NEIL P，O'NEIL B，CHEN X D. Star Schema Benchmark Revision[EB/OL]. [2010-2-9]. http：//www. cs. umb. edu/~poneil.