软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(8):2032−2041 [doi: 10.3724/SP.J.1001.2012.04124]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 Hadoop 的高效连接查询处理算法 CHMJ*

赵彦荣[1,2,3+], 王伟平[1,2], 孟 丹[1,2], 张书彬[4], 李 均[4]

[1](中国科学院 计算技术研究所,北京 100190)

[2](中国科学院 国家智能计算机研究开发中心,北京 100190)

[3](中国科学院 研究生院,北京 100049)

[4](腾讯公司 数据平台部,广东 深圳 518057)

Efficient Join Query Processing Algorithm CHMJ Based on Hadoop

ZHAO Yan-Rong[1,2,3+], WANG Wei-Ping[1,2], MENG Dan[1,2], ZHANG Shu-Bin[4], LI Jun[4]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](National Research Center for Intelligent Computing Systems, The Chinese Academy of Sciences, Beijing 100190, China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[4](Data Platform Department, Tencent, Inc., Shenzhen 518057, China)

+ Corresponding author: E-mail: zhaoyanrong@ncic.ac.cn

Zhao YR, Wang WP, Meng D, Zhang SB, Li J. Efficient Join query processing algorithm CHMJ based on Hadoop. *Journal of Software*, 2012,23(8):2032−2041 (in Chinese). http://www.jos.org.cn/1000-9825/4124.htm

Abstract: This paper proposes a join query processing algorithm CoLocationHashMapJoin (CHMJ). First the study designs a multi-copy consistency hash algorithm. The algorithm distributes the data of tables over the cluster according to the hash values of the join property, which improves the data locality while ensure data availability. Second, based on the multi-copy consistency hash algorithm, the study proposes a parallel join query processing algorithm called HashMapJoin. HashMapJoin improves the efficiency of join query significantly. CHMJ has been used in Tencent's data warehouse system, and plays an important role in Tencent's daily analysis tasks. The results show that CHMJ improves the efficiency of join query processing by five times comparing to Hive.

Key words: big data; Hadoop; join query processing; HashMapJoin

摘 要: 提出了一种并行连接查询处理算法 CoLocationHashMapJoin(CHMJ).首先,设计了多副本一致性哈希算法,将具有连接关系的表根据其连接属性的哈希值在机群中进行分布,在提升了连接查询处理中数据本地性的同时,保证了数据的可用性;其次,基于多副本一致性哈希数据分布,提出了 HashMapJoin 并行连接查询处理算法,有效地提高了连接查询的处理效率.CHMJ 算法在腾讯公司的数据仓库系统中进行了应用,结果表明,CHMJ 连接查询的处理效率比 Hive 系统提高了近 5 倍.

关键词: 大数据;Hadoop;连接查询处理;HashMapJoin

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家自然科学基金(60903047)

收稿时间: 2011-05-12; 定稿时间: 2011-09-01

随着互联网应用的快速发展,海量数据的存储与处理成为研究人员面临的严峻挑战.传统的数据库技术由于可扩展能力不强、成本较高等方面的限制,无法满足海量数据管理的需求.近年来,以谷歌公司提出的分布式文件系统 GFS[1]、并行编程框架 MapReduce[2−4]为代表的技术成为海量数据存储与分析处理的主流技术.基于 GFS 和 MapReduce 的设计思想,开源社区 Apache 的 Hadoop 项目实现了分布式文件系统 Hadoop DFS 和并行编程框架 Hadoop MapReduce.目前,Hadoop 广泛地应用在雅虎、脸谱等互联网公司的海量数据存储与分析应用中.由于 MapReduce 编程模型处于较低层次,针对不同的数据分析任务,开发人员需要编写不同的 MapReduce 程序进行处理,程序难以维护与重用.为了方便上层应用的开发,Hive[5−7],Pig[8]等技术被提出来,对 MapReduce 编程框架进行了封装,为上层应用提供更易使用的类 SQL 调用接口.

在统计分析类的查询中,连接(join)是主要的查询操作之一.Hive 在处理连接查询时采用了排序归并算法(SortMergeReduceJoin,以下简称 Reduce Join).该算法的执行分为 Map 和 Reduce 两个阶段:Map 阶段对执行连接的两个表在连接属性上进行分段排序;Reduce 阶段将各个 Map 阶段生成的分段排序结果进行归并连接,输出查询结果.该算法存在两个问题:(1) Map 阶段产生的大量中间结果数据需要通过网络传输到 Reduce 端,消耗了大量的带宽,降低了算法执行的效率;(2) Reduce 端需要进行多次归并排序操作,因而开销较大,执行时间较长.

针对上述问题,本文提出了一种基于数据本地化计算的连接查询处理算法 CoLocationHashMapJoin(以下简称 CHMJ).CHMJ 算法的基本思想是,将经常做连接的两个数据表根据其连接属性的值在机群中进行哈希分布,在处理连接查询时,CHMJ 算法只需在 Map 阶段执行 HashJoin 算法即可得到连接结果,无须执行 Reduce 阶段,从而降低了查询处理的时间开销.本文提出的算法在腾讯公司数据仓库(tencent distributed data warehouse,简称 TDW)中进行了应用,结果表明,CHMJ 算法在处理分区属性上的连接查询时,执行时间仅为 Hive 的 Reduce Join 算法的 20%左右.本文提出的思想同样适用于分组(groupby)查询的处理.

本文第 1 节介绍相关工作.第 2 节介绍 CHMJ 算法的主要思想,并对算法的性能进行分析.第 3 节给出实验结果与分析.第 4 节对全文进行总结.

1 相关工作

在海量数据的存储与处理方面,由于有大量的科研院所和大公司的推动,技术发展非常迅速.谷歌、亚马逊、IBM 和微软等公司都在此领域投入了大量的科研力量,提出了多种创新的海量数据管理技术.这些研究工作主要集中在海量数据存储、海量数据计算和用户接口这 3 个层面.

存储层提供海量数据的可靠存储与高效文件访问服务,主要技术包括谷歌公司的分布式文件系统 GFS、Hadoop 的 HDFS、KFS[9,10]、亚马逊公司的 S3[11,12]等.计算层提供对海量数据的并行计算服务,计算层通过存储层提供的 API 读取数据,在执行过程中会将作业实时运行状态返回给接口层,直到作业运行结束,并将最终作业执行结果反馈给接口层.面向海量数据处理的并行计算技术包括谷歌公司的 MapReduce,Hadoop MapReduce 和微软的 Dryad[13]等.接口层的功能是对外提供编程 API、交互式 shell 以及 Web GUI 等界面,用户可以向系统提交类 SQL 语句(或其他查询语言),接口层对 SQL 语句进行语法分析、语义分析和查询优化,最后将其转化为 1 个或多个作业,调度计算层执行.目前,主要的接口层研究工作包括 Hive,Pig,Scope[14],DryadLINQ[15],Sawzall[16]等.

Hive 是接口层的一个代表性工作,主要应用在脸谱公司的海量数据分析处理中.Hive 以 Hadoop DFS 作为存储引擎,以 Hadoop MapReduce 作为计算引擎,对上层应用提供类 SQL 的调用接口.Hive 处理连接查询时,主要采用 Reduce Join 算法.

如图 1 所示,Reduce Join 算法的执行分为 3 个阶段:Map 阶段、Shuffle/Sort 阶段和 Reduce 阶段.在 Map 阶段,每个 Map 任务处理一个数据块,Map 任务输出的临时结果以〈key,value〉对的形式写入本地,将连接属性的值作为键值,并在 value 中打入表的标签以表明数据的来源,Map 任务输出的临时结果通过主键值进行分区.在 Shuffle/Sort 阶段,Map 任务的临时结果将按键值传输到相应的 Reduce 任务.Reduce 任务会聚集来自多个 Map 任务端的分区数据,执行多次 Sort/Merge 操作,将分区数据进行合并与排序,通过 Reduce 函数进行连接,并将最

终的连接查询结果输出.

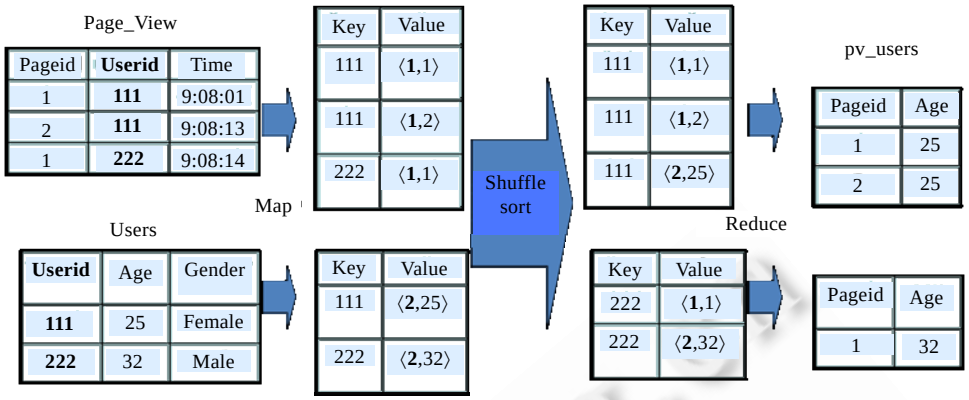

Fig.1 Process of SortMergeReduceJoin

图 1 SortMergeReduceJoin 流程

在 Reduce Join 的执行过程中,只有在所有 Map 阶段完成后,Reduce 阶段才能执行.Reduce 任务需要拉取 Map 端的中间结果数据,带来了大量的网络通信负载,效率较低.同时,Reduce 任务需要对收到的数据多次执行 Sort/Merge 操作,也需要大量的计算,算法执行效率较低.

2 CHMJ 连接查询处理算法

针对 Reduce Join 执行效率较低的问题,提出了一种基于数据本地化计算的连接查询处理技术 CHMJ.该技术包含两部分:数据分布策略和连接查询处理算法 HashMapJoin.

数据分布对表的特定属性进行哈希,将表中的数据划分到相应哈希分区之中.数据分布策略将逻辑上相关的数据,聚集存储在相同节点上,使得属于同一个哈希分区的数据以及具有相同哈希分区序号但属于不同表的数据能够集中存放在同一个节点.

由于采用哈希技术将数据表在机群上进行了分布,HashMapJoin 算法只需执行 Map 任务,以分区作为处理单位,进行连接查询处理并直接完成结果输出,从而避免了 Reduce 操作带来的巨大时间开销,大幅度提升了连接查询处理的效率.

2.1 数据分布策略

在并行查询处理中,数据的分布策略是一个关键问题,直接影响查询处理的执行效率.目前,围绕在数据分布策略方面已有大量的研究工作,提出了许多有效的并行数据分布方法,例如 Round-Robin,Hash,Range-Partition[17],CMD[18]以及基于聚类和一致 Hash 的数据布局[19]等数据分布方法.

CHMJ 采用哈希技术对数据进行分布,对表的某个属性进行哈希,根据哈希值将数据划分到相应哈希分区之中.就连接查询而言,通过哈希分区,可以将连接查询转换为 Map 端进行的计算,而且通过控制哈希分区的数目可以控制连接查询的粒度,进一步优化对计算资源的利用.哈希分区只是实现了逻辑上的相关数据聚合,在物理层次上,一个分区往往包含多个文件,每一个文件又由多个数据块构成,这些数据块则会分布在不同的节点上.哈希数据分布策略的目的是将这些相关的数据集中存放在同一个物理节点上.同时,数据分布策略也提供对列存储的支持,对列存储文件而言,同一元组的不同列或列簇能够被聚集存放在相同节点,可以进一步发挥列存储的优势.

CHMJ 以分布式文件系统 Hadoop DFS 作为底层存储,而 Hadoop DFS 不支持应用感知的数据分布方法.我们在 Hadoop DFS 基础上增加了哈希数据分布策略.针对 Hadoop DFS 采用数据多副本的容错机制,我们提出了多副本一致性哈希算法,在支持数据本地化计算同时,也保证了修改后的 Hadoop DFS 的容错能力不变.多副本

一致性哈希算法基于一致性哈希算法,一致性哈希算法[20]的思想主要是为了解决网络中的热点问题.鉴于一致性哈希算法只考虑单个数据没有考虑多个副本的问题,多副本一致性哈希算法修正与扩充了一致性哈希算法,其支持多个副本的存放方式,从而达到多个副本的相关数据同样达到聚集的效果.多副本一致性哈希算法同时也支持 Dynamo 中提出的虚拟节点的概念[21],一个实节点可以影射为多个虚拟节点.在多副本一致性哈希算法中需要构建一个哈希环,哈希环配置如图 2 所示.

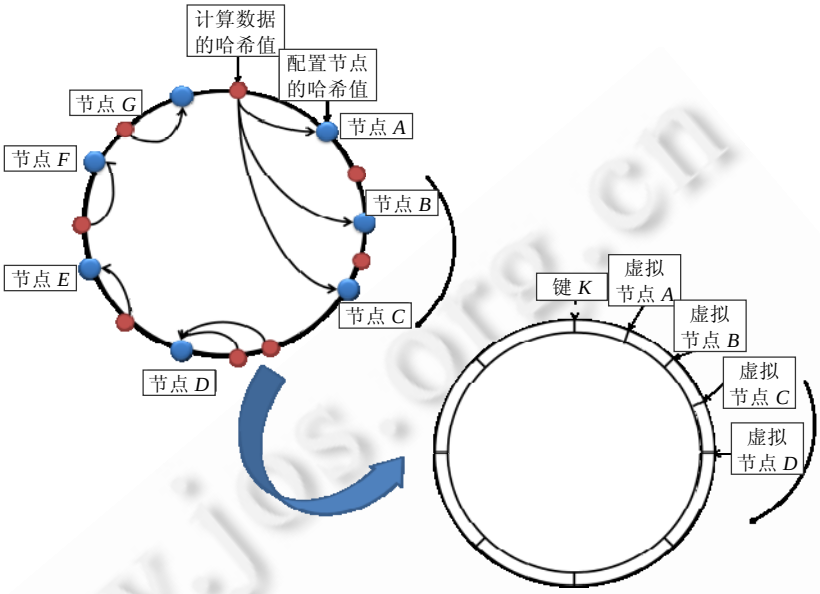

Fig.2 Example of hash ring

图 2 Hash 环示例

本文提出的多副本一致性哈希算法如下:

(1) 对节点首先求出数据节点的哈希值,将其配置到一个 Hash 环上.

(2) 计算数据的哈希值(对于哈希分区而言,哈希分区的值就是数据的哈希值;对于非哈希分区的列存储文件或者列簇式存储文件,哈希值是其在存储文件系统中的路径名去除列标签后进行哈希后得到的值),也将其映射到哈希环上.

(3) 按顺时针将数据映射到距离其最近的节点上.对于多副本数据,则顺环继续选取新的节点,直到有足够的副本存放节点,映射到的节点保存该数据.

(4) 若选取的节点为虚节点,且该虚节点所表示的实际节点已经存放了一个副本,则跳过该节点继续寻找下一个节点.若节点出现磁盘满,或者节点异常等状况,则跳过该节点继续寻找下一个节点.

多副本一致哈希算法可以避免节点频繁加入退出、改动原有映射关系所带来的巨大开销,同时避免了归属于同一哈希分区的数据或者同一个表的不同列文件被映射到不同的节点,相关数据可以实现物理层次的聚集,从而提升查询时的性能.

2.2 HashMapJoin连接查询处理算法

HashMapJoin 是一个在 Map 端执行的并行连接查询处理算法.算法主要思想是,针对用户提交的连接查询,通过判断参与计算的多个数据表均进行了哈希分区且哈希分区数相同,从而判断出该查询可以由 HashMapJoin 算法进行处理.查询处理引擎会对用户提交的 SQL 语句进行相应的词法分析、语法分析,并产生优化后的 MapReduce 任务.优化后的 Map 任务将以 Hash 分区作为计算单位,Hash 分区的数量最终将生成相应个 Map 端的连接查询任务.HashMapJoin 算法仅有 Map 阶段而无 Reduce 阶段,查询处理的结果将直接通过 Map 端输出.

如图 3 所示,数据表 *a* 和数据表 *b* 的同一个哈希分区 Hash-XXXX 将由同一个 Map 计算任务进行连接计算.所有这些哈希分区的计算结果汇总起来就是正确的连接查询结果,各个任务之间无须交互.利用数据多副本一致性哈希,将相关的哈希分区的数据存放于同一个物理节点上,可节省网络带宽和计算资源.

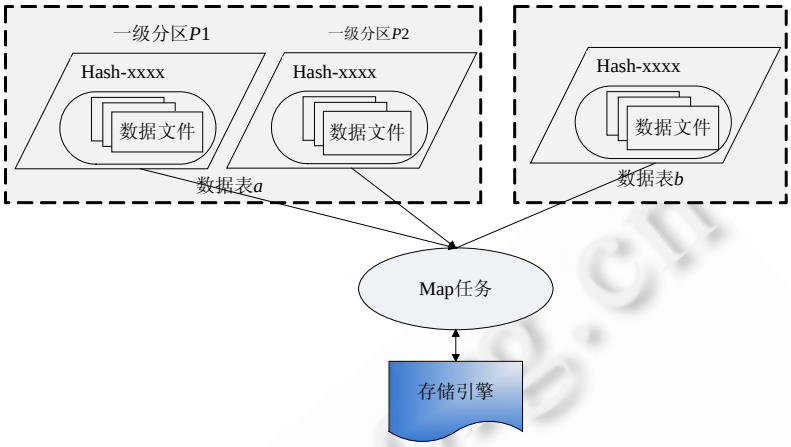

Fig.3 HashMapJoin query of two tables

图 3 两个表的 HashMapJoin 查询

2.3 算法性能分析

CHMJ 算法的主要目的是充分利用数据的局部性,避免查询处理时的网络通信,以提升连接查询处理的效率.下面分析在没有采用多副本一致性哈希算法分布数据的情况下,HashMapJoin 算法的执行效率.假设集群节点数为 M;B_j 为任务 j 计算所涉及的数据块总数,$b_i(0\leqslant i<M)$,代表节点 i 包含的任务涉及的数据块数量,则 $B_j=\sum_{i=0}^{M-1}b_i$;C 为 Hash 分区数,D 为数据总量,则每个数据块的平均数据量为 $\frac{D}{B_j}$,任务 j 中相同分区计算时所涉及的平均数据块数为 $\frac{B_j}{C}$.Map 以分区作为处理单元时,调度算法会尽量保证一个数据块的数据本地性,则每个 Map 其余的数据块个数为 $\frac{B_j}{C}-1$,每个 Map 的数据本地性概率为 $\left(\frac{1}{M}\right)^{\left|\frac{B_j}{C}-1\right|}$.假设有 R 个副本,则每个 Map 的数据本地性概率为 $\left(\frac{R}{M}\right)^{\left|\frac{B_j}{C}-1\right|}$,则总体的数据完全本地性概率为 $\left(\frac{R}{M}\right)^{\left|\frac{B_j}{C}-1\right|\times C}$.在 HashMapJoin 的执行中,由于任务调度会尽量保证一个数据块的数据本地性,使得每个 Map 需要拉取的数据量为

$$\left|\frac{B_j}{C}-1\right|\times\left[\left(1-\frac{R}{M}\right)\times\frac{D}{B_j}\right]=\left|\frac{B_j}{C}-1\right|\times\left[\left(1-\frac{R}{M}\right)\times\frac{D}{\sum_{i=0}^{M-1}b_i}\right].$$

总的数据拉取量 P 为

$$P=\left|\frac{B_j}{C}-1\right|\times\left[\left(1-\frac{R}{M}\right)\times\frac{D}{\sum_{i=0}^{M-1}b_i}\right]\times C \tag{1}$$

假设 HashMapJoin 任务处理的数据总量 D 为 3TB,Hash 分区数 C 为 500,任务 j 计算所涉及的数据块总数为 1 000,即 B_j 为 10 000,则在 HashMapJoin 中,每个 Map 平均处理的数据块数,即 B_j/C 为 20,机器数量范围为

[3,250],可以得到图 4.

由图 4 可知,首先,网络 I/O 的负载量很重,随着节点数量 M 的扩大,拉取的数据量呈单调递增态势,数据本地性越差,算法性能越差;其次,副本数量对 Map 拉取的数据量也有影响,当机群节点数量在较小范围时,副本数量对拉取数据量影响差别很大,但随着机器数量的增长,差异逐步缩小.

当机群节点数量非常小,为 3,并且数据副本数量同样为 3 时,则每个节点都完全拥有所有数据的一个完整副本集.由图 4 可以看出:其不需要从网络上拉取数据,而当副本数量分别为 1,2 时,则分别需要拉取 1.9TB,0.95TB 数据.当机群节点数量达到一定小规模,如 50 台机器时,副本数量分别为 1,2,3 时,分别需要拉取数据 2.79TB,2.736TB,2.679TB,分别占总计算数据量的 93%,91.2%,89.3%.当机群节点数达到较大规模如 250 台时,则分别拉取 2.839TB,2.8272TB,2.816TB,分别占总计算数据量的 94.6%,94.3%,93.7%.

由此可见,在没有采用 CoLocation 的情况下,数据本地性很差,仅有非常少量的数据能够达到本地性,大量数据都需要经过网络 I/O 传输.

由于存储层对于文件的存储是按块存储的.在处理的数据总量不变的情况下,调整数据块的大小,假设每个 Map 平均处理的数据块数 B_j/C 分别为 10,20,30 时,数据副本数均为 3,其余参数不变,得到图 5.

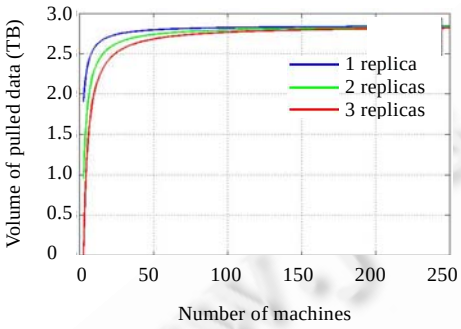

Fig.4 Traffic of different number of data copies

图 4 不同副本数量下的数据流量

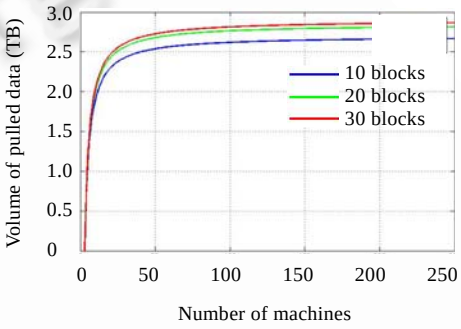

Fig.5 Traffic of different block size of data

图 5 不同数据块数量下的数据流量

由图 5 可以看出,当处理数据总量不变的情况下,数据块的大小同样对拉取数据量有一定的影响,配置较大的数据块可以在一定程度上减少网络数据的拉取量,提高数据的本地性.但就总体而言,数据拉取量仍很大.

综上,在没有采用多副本一致性哈希算法分布数据的情况下,执行 HashMapJoin 时,数据本地性会较差;即使在较小的机群数量状况下,也仅有非常少量数据能够达到本地性,绝大部分数据都需要经过网络传输,网络负载很重.如果采用多副本一致性哈希算法分布数据,则相关数据都被聚集在一个物理节点之上,通过将处理任务调度到数据聚集存放的物理节点,可以使数据的本地性达到 100%,可以大幅度降低大集群系统中的网络负载,提升连接查询的处理性能.

3 实验与结果分析

3.1 实验环境与数据

为了测试 CHMJ 算法的相关性能,我们将其与目前最好的 Hive 系统的 SortMergeReduceJoin(Reduce Join)连接查询处理算法做了对比.实验环境为腾讯公司 30 台服务器,每台服务器的配置为:8 个处理核心,16GB 内存,12 块硬盘,每块硬盘容量为 1TB.其中,一个节点作为主控节点,一个节点作为客户端,另外 28 个节点作为存储与计算节点,在测试中仅使用各个节点的一块硬盘.实验所用的数据表在入库时根据腾讯公司的 qq 号码属性划分为 500 个 Hash 分区,数据块大小设置为 128MB,网络带宽为各节点独享 100Mb,每个计算节点分配 6 个 Map 计算槽、2 个 Reduce 计算槽.实验使用了腾讯公司两个较大的真实数据集,数据集的具体情况见表 1.

Table 1 Dataset used in the experiments

表 1 实验数据集

ata table	F plication	Data information		Record number (Billion)
		Data ize (GB)	Data occupied ize (GB)	
userprofile	3	15.4	46.2	0.6
qqfriend	3	520	1 560	12.7

3.2 实验与分析

我们对 Hive 系统的 Reduce Join,HashMapJoin 与 CHMJ 进行对比性能测试与分析,具体实验结果如下:

3.2.1 全表连接查询的性能对比实验

实验 1 对全表连接查询的处理性能进行了对比分析.实验中采用的查询语见表 2.其中,Reduce Join 采用 H-SQL,而 HashMapJoin 和 CHMJ 采用了腾讯公司数据仓库中的查询语言 Tencent-SQL.这些查询语句的语义是一致的.

Table 2 SQL statements used in the Experiment 1

表 2 实验 1 中的 SQL 语句

Types of join	SQL statements
Reduce Join	insert overwrite table tmp select a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1
HashMapJoin	insert overwrite table tmp select /*+hashmapjoin(a)*/ a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1
CHMJ	insert overwrite table tmp select /*+hashmapjoin(a)*/ a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1

实验结果如图 6 所示,HashMapJoin 性能远优于 Hive 的 Reduce Join,前者的执行时间仅为后者的 49.8%.造成这种结果的主要原因在于:1) Reduce 任务的启动开销较大;2) Map 与 Reduce 任务需要从网络拉取数据,这其中不仅涉及网络开销,而且需要进行多次硬盘 I/O.而考虑数据本地性的 CHMJ 比 Reduce Join 较 HashMapJoin 有更一步的性能提升,执行时间仅为 Hive 的 Reduce Join 的 26%,加速比为 3.76.

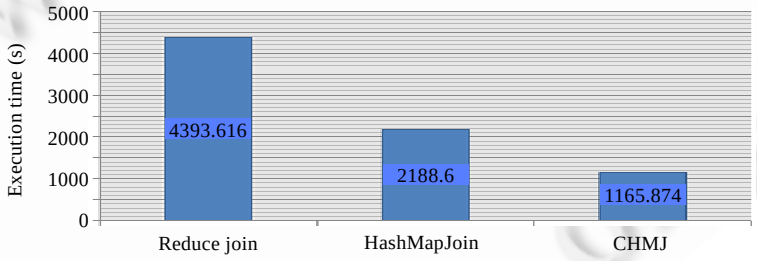

Fig.6 Comparison of execution time of Reduce Join, HashMapJoin and CHMJ

图 6 Reduce Join,HashMapJoin 和 CHMJ 的执行时间对比

3.2.2 带有区间范围的连接查询性能对比实验

实验 2 测试了带有区间范围的连接查询处理性能.选择性查询条件为 QQ 号小于 200 000 000,Reduce Join 采用 H-SQL 表达,HashMapJoin 和 CHMJ 采用 Tencent-SQL 表达.查询语句见表 3.

实验结果如图 7 所示,HashMapJoin 相对于 Reduce Join 的加速比为 2.06,CHMJ 相对于 Reduce Join 的加速比为 4.63.带有区间范围的 CHMJ 连接查询相对于全表 CHMJ 连接查询,其结果输出量较少,受网络状况制约较少,因而加速比更大.

Table 3 SQL statements used in the Experiment 2

表 3 实验 2 中的 SQL 语句

Types of join	SQL statements
Reduce Join	insert overwrite table tmp select a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1 where a.QQ_Num<200000000
HashMapJoin	insert overwrite table tmp select /*+hashmapjoin(a)*/ a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1 where a.QQ_Num<200000000
CHMJ	insert overwrite table tmp select /*+hashmapjoin(a)*/ a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1 where a.QQ_Num<200000000

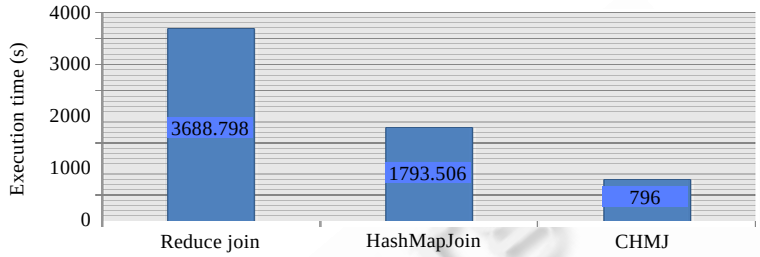

Fig.7 Comparison of execution time of Reduce Join, HashMapJoin and CHMJ

图 7 Reduce Join,HashMapJoin 与 CHMJ 执行时间对比

3.2.3 低选择性连接查询的性能对比实验

实验 3 验证了带有低选择性查询条件的各个算法性能.在实验 3 中,低选择性的查询条件为 a.QQ_Num=126 357,Reduce Join 采用 H-SQL 语言表达,HashMapJoin 和 CHMJ 采用 Tencent-SQL 语言表达.具体的查询语句见表 4.

Table 4 SQL statements used in the Experiment 3

表 4 实验 3 中的 SQL 语句

Types of join	SQL statements
Reduce Join	insert overwrite table tmp select a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1 where a.QQ_Num=126357
HashMapJoin	insert overwrite table tmp select /*+hashmapjoin(a)*/ a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1 where a.QQ_Num=126357
CHMJ	insert overwrite table tmp select /*+hashmapjoin(a)*/ a.STATIS_MONTH, b.qq1, a.AGE, a.GENDER, b.qq2 from userprofile a join qqfriend b on a.QQ_NUM=b.qq1 where a.QQ_Num=126357

实验结果如图 8 所示,HashMapJoin 相对于 Reduce Join 的加速比为 2.49,CHMJ 仅是 Reduce Join 执行时间的 19.2%,加速比为 5.2.实验结果表明,CHMJ 算法性能远优于 Hive 的 Reduce Join.相对于全表连接查询,实验 3 特点在于大量数据输入,极少量数据结果输出.就 CHMJ 算法而言,其数据均来自于本地,且极少量结果输出,受网络状况影响非常小,获得的加速比最大,性能优势更为明显.

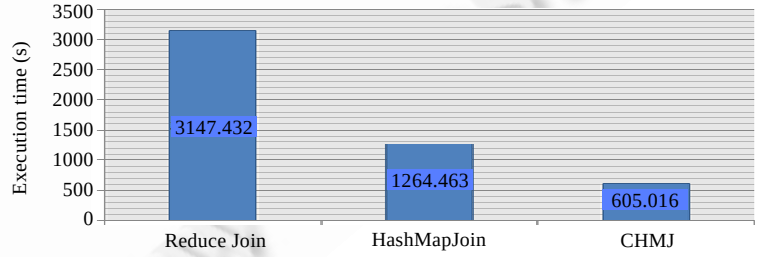

Fig.8 Comparison of execution time of Reduce Join, HashMapJoin and CHMJ

图 8 Reduce Join,HashMapJoin 和 CHMJ 执行时间对比

3.2.4 负载均衡性实验

实验 4 测试了数据本地化计算分布策略对数据负载均衡性造成的影响,并与 Hadoop 默认机架感知分布策

略进行对比.实验中采用 28 台服务器作为存储节点,数据表以腾讯 QQ 号码进行划分,Hash 分区数量为 500,实验结果如图 9 所示.

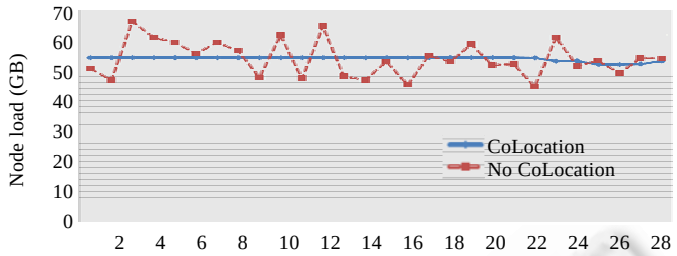

Fig.9 Cluster nodes load

图 9 集群节点负载

如图 9 所示,采用数据本地化计算策略存放的数据的负载状况呈线状,而在机架感知分布策略下的负载状况呈锯齿状.由于副本数为 3,根据多副本一致性哈希算法,6 个节点存储的哈希分区数量少于其余 22 个节点.

由 $\bar{X}=\frac{1}{n}\sum_{i=1}^{n}X_i$ 可以得到负载的平均值,并使用样本方差 $S^2=\frac{1}{n-1}\sum_{i=1}^{n}(X_i-\bar{X})^2$ 来衡量负载的波动性.通过定量分析,得到采用数据本地化计算策略的样本方差为 0.00064,机架感知分布策略的样本方差为 40.59.本地化计算策略的负载波动远小于机架感知分布策略.相对于机架感知分布策略随机选取存放节点造成的数据负载不均,采用数据本地化计算策略可以改善集群的负载分配状况.

4 结 论

本文提出了一种高效的并行连接查询处理算法 CHMJ.CHMJ 设计了多副本一致性哈希算法,将具有连接关系的表根据其连接属性的哈希值在机群中进行分布,在提升连接查询处理中数据本地性的同时,也保证了数据的可用性.在多副本一致性哈希数据分布的基础上,提出了 HashMapJoin 并行连接查询处理算法,有效地提高了连接查询的处理效率.CHMJ 算法在腾讯公司的数据仓库系统中进行了应用,结果表明,CHMJ 连接查询的处理效率比 Hive 系统提高了近 5 倍.

致谢 在此,我们向对本文工作提出建议和意见的评审专家及腾讯公司 TDW 系统研发小组的同事表示感谢.

References:

[1] Ghemawat S, Gobioff H, Leung ST. The Google file system. In: Proc. of the SOSP 2003. 2003. 20−43. [doi: 10.1145/1165389.945450]

[2] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In: Proc. of the OSDI 2004. 2004. 137−150. [doi: 10.1145/1327452.1327492]

[3] Yang HC, Dasdan A, Hsiao RL, Parker DS. Map-Reduce-Merge: Simplified relational data processing on large cluster. In: Proc. of the SIGMOD 2007. 2007. 1029−1040. [doi: 10.1145/1247480.1247602]

[4] Lämmel R. Google's MapReduce programming model—Revisited. Science Computer Program, 2008,70(1):1−30. [doi: 10.1016/j.scico.2007.07.001]

[5] Apache Hive. http://hadoop.apache.org/hive/

[6] Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R. Hive: A warehousing solution over a map-reduce framework. Proc. of the VLDB Endowment, 2009,2(2):1626−1627.

[7] Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Antony S, Liu H, Murthy R. Hive—A petabyte scale data warehouse using Hadoop data engineering. In: Proc. of the ICDE. 2010. 996−1005. [doi: 10.1109/ICDE.2010.5447738]

[8] Olston C, Reed B, Sirvastava U, Kumar R, Tomkins A. Pig Latin: A not-so-foreign language for data processing. In: Proc. of the SIGMOD. 2008. 1099−1110. [doi: 10.1145/1376616.1376726]

[9] White T. Hadoop: The Definitive Guide. O'Reilly, 2009.

[10] Apache Hadoop. http://hadoop.apache.org/

[11] Murty J. Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB. O'Reilly, 2008.

[12] Patten S. The S3 Sookbook: Get cooking with Amazon's Simple Storage Service. Sopobo, 2009.

[13] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed data-parallel programs from sequential building blocks. In: Proc. of the EuroSys 2007. 2007. 59−72. [doi: 10.1145/1272998.1273005]

[14] Chaiken R, Jenkins B, Larson PA, Ramsey B, Shakib D, Weaver S, Zhou JR. SCOPE: Easy and efficient parallel processing of massive data sets. Proc. of the VLDB Endowment, 2008,1(2):1265−1276. [doi: 10.1145/1454159.1454166]

[15] Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson U, Gunda PK, Currey J. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In: Proc. of the OSDI. 2008. 1−14. [doi: 10.1145/1559845.1559962]

[16] Pike R, Dorward S, Griesemer R, Quinlan S. Interpreting the data: Parallel analysis with sawzall. Scientific Programming Journal, 2005,13(4):227−298.

[17] De Witt DJ, Gerber RH, Graefe G, Heytens ML, Kumar KB, Muralikrishna M. GAMMA: A performance dataflow database machine. In: Proc. of the Int'l Conf. on VLDB. 1986. 228−237.

[18] Li JZ, Srivastava J, Rotem D. CMD: A multidimensional declustering method for parallel database system. In: Proc. of the 18th VLDB Conf. 1992. 310−327.

[19] Chen T, Xiao N, Liu F, Fu CS. Clustering-Based and consistent hashing-aware data placement algorithm. Journal of Software, 2010,21(12):3175−3185 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3706.htm [doi: 10.3724/SP.J.1001.2010.03706]

[20] Karger D, Lehman E, Leighton T, Levine M, Lewin D, Panigrahy R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In: Proc. of the STOC'97. 1997. 654−663. [doi: 10.1145/258533.258660]

[21] Hastorun D, Jampani M, Kakulapati G, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. In: Proc. of the SOSP 2007. 2007. 205−220. [doi: 10.1145/1323293.1294281]

附中文参考文献:

[19] 陈涛,萧侬,刘芳,付长胜.基于聚类和一致 Hash 的数据布局算法.软件学报,2010,21(12):3175−3185. http://www.jos.org.cn/1000-9825/3706.htm [doi: 10.3724/SP.J.1001.2010.03706]

赵彦荣(1983－),男,陕西西安人,博士,CCF 学生会员,主要研究领域为网络,分布式数据仓库,云计算.

张书彬(1979－),男,博士,工程师,主要研究领域为分布式数据仓库,云计算.

王伟平(1975－),男,博士,副研究员,CCF 会员,主要研究领域为数据库.

李均(1978－),男,工程师,主要研究领域为海量数据处理.

孟丹(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为计算机系统结构.