文章编号:0253-2778(2015)10-0836-10

PipelineJoin:一种新的基于MapReduce的多表连接算法

林子雨[1],李雨倩[1],李 粲[1],赖永炫[2]

(厦门大学信息科学与技术学院,福建 厦门 361005;2. 厦门大学软件学院,福建 厦门 361005)

摘要:MapReduce是一个并行分布式计算模型,已经被广泛应用于处理两个或多个大型表的连接操作.现有的基于MapReduce的多表连接算法,在处理链式连接时,不能处理多个大表的连接,或者需要顺序运行较多的MapReduce任务,效率较低.为此提出了一种基于MapReduce的多表连接算法——PipelineJoin,高效地实现任意多个大表的链式连接.PipelineJoin采用流水线模型和调度器来实现MapReduce任务的流水线式执行,从而有效提高多表连接的效率,同时可以较好地克服链式多表连接算法的缺陷.最后,在不同规模的数据集上进行了大量实验,实验结果表明PipelineJoin算法与原有链式多表连接算法相比,可以有效减少连接所需的时间.

关键词:连接;多表;MapReduce;PipelineJoin

中图分类号:TP18 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2015.10.006

引用格式:LIN Ziyu, LI Yuqian, LI Can, et al. PipelineJoin:A new MapReduce-based multi-table join algorithm[J]. Journal of University of Science and Technology of China, 2015,45(10):836-845.
林子雨,李雨倩,李 粲,等. PipelineJoin:一种新的基于MapReduce的多表连接算法[J]. 中国科学技术大学学报,2015,45(10):836-845.

PipelineJoin:A new MapReduce-based multi-table join algorithm

LIN Ziyu[1], LI Yuqian[1], LI Can[1], LAI Yongxuan[2]

(1. *College of Information Science and Technology, Xiamen University, Xiamen* 361005, *China*;
2. *School of Software, Xiamen University, Xiamen* 361005, *China*)

Abstract: MapReduce, a parallel and distributed computing model, has been widely used to process join operations for two or more large tables. The existing MapReduce-based multi-table join algorithms all have some limitations when dealing with chain join. Some methods can not process join operations for multi large tables, and others involve sequentially running too many MapReduce tasks, which leads to low efficiency. Here a new MapReduce-based multi-table join algorithm, PipelineJoin, is proposed to process chain join of a number of tables. PipelineJoin adopts a pipeline model and a scheduler to allow the overlapping execution of a series of Map tasks and Reduce tasks in the whole join process so as to enhance the efficiency of multi-table join, while effectively overcoming the deficiency of the existing methods. Extensive experimental results based on various synthetic datasets show that the proposed algorithm can greatly reduce join operation time compared with the existing chain join algorithms.

Key words: join; multi-table; MapReduce; PipelineJoin

收稿日期:2015-08-27;**修回日期**:2015-09-29

基金项目:国家自然科学基金(61303004,1202012);国家科技支撑计划(863)(2015BAH16F00/F01/F02)资助.

作者简介:林子雨(通讯作者),男,1987年生,博士/讲师,研究方向:大数据挖掘. E-mail:ziylin@xmu.edu.cn

0　引言

大数据时代的到来，带动了数据量的迅猛增长，急需一种技术来存储和处理如此庞大的数据量，由此，谷歌的分布式文件系统 GFS 和分布式计算模型 MapReduce 应运而生. Hadoop 开源实现了谷歌的 GFS 和 MapReduce，它提供了一个能够让用户轻松架构和使用的分布式计算平台，用户可以轻松地在 Hadoop 平台上开发和运行处理海量数据的应用程序. Hadoop 主要由 HDFS（Hadoop 分布式文件系统）和 MapReduce 两部分组成. 其中，HDFS 提供了海量数据的存储能力，MapReduce 提供了海量数据的处理能力. MapReduce 已经得到了广泛的应用，并被用于处理数据库中两个或多个表的连接操作. 一般来说，多表间的连接形式主要分为三种：星型连接、链式连接以及两者的混合. 其中，星型连接实现的是一个中心表和多个与该中心表至少有一个共同属性的表之间的连接. 链式连接实现的是多个两两间具有相同属性的多个表之间的连接. 本文研究的是链式多表连接的情况. 它的具体形式表示如下：

$$T_1(A_0,A_1)\infty T_2(A_1,A_2)\infty T_3(A_2,A_3)\infty\cdots\infty T_n(A_{n-1},A_n) \tag{1}$$

式中，T_1、T_2、…、T_n 表示关系，A_1、A_2、…、A_n 表示关系中的属性. 目前，基于 MapReduce 链式多表连接的算法主要分为两种：级联连接和多路连接.

级联连接采用多个连续的二路连接处理多表连接问题. 具体来说，级联连接先执行前两个表的连接操作，接着把连接的结果看作一个新的表，再和下一个表进行连接，以此类推，直到完成所有表的连接. 级联连接的优势是可以处理任意数量、任意大小的表，只要它可以存储在 HDFS 上即可. 例如，对于待连接的 n 个表 T_1、T_2、T_3、…、T_n（相邻两个表存在连接键），级联连接先运行一个 MapReduce 任务处理 T_1 和 T_2 的连接，然后再运行一个 MapReduce 任务把 T_1、T_2 的连接结果和 T_3 做连接，以此类推，直到表 T_n 完成连接为止. 用关系代数式表示如下：

$$((\cdots((T_1\infty T_2)\infty T_3)\infty\cdots)\infty T_n) \tag{2}$$

级联连接存在一个显著的缺点，即需要运行多个 MapReduce 任务才能完成连接操作，效率较低.

多路连接可以只运行一个 MapReduce 任务就完成这 n 个表的连接操作，它的关系代数式表示如下：

$$T_1\infty T_2\infty T_3\infty\cdots\infty T_n \tag{3}$$

多路连接采用哈希桶的方式，在 Map 端把某个 Reduce 端连接需要的所有数据发送给该 Reduce 任务. 当三个表 T_1、T_2 和 T_3 连接时，对于接收了属于 T_2 的数据块 D_1 的 Reduce 任务而言，它还需接收所有需要和 D_1 做连接的 T_1 和 T_3 中的数据. 当有四个表 T_1、T_2、T_3 和 T_4 时，对于同时接收了属于 T_2 和 T_3 的数据块 D_2 的 Reduce 任务而言，它还需接收所有需要和 D_2 做连接的 T_1 和 T_4 中的数据. 与三表连接相比，四表连接需要的 Reduce 机器数和中间数据传输量呈线性增加；与级联连接相比，多路连接减少了需要开启的 MapReduce 任务数和存储中间结果的代价. 多路连接存在一个明显的不足，即需要传输给 Reduce 机器的数据量和需要开启 Reduce 机器的个数会随着表的增多呈线性增加.

文献[1]提出的 SmartJoin 算法，采用比级联连接更少的 MapReduce 任务数来实现多路连接不能实现的大量表之间的连接操作，但是它对表有太多限制. 首先，它要求参与连接的表必须是两个大表和可以同时存储在一个 Reduce 机器下的多个小表；其次，它还要求这两个大表之间必须存在连接键，但是对于链式多表连接来说，无法保证这两个大表之间一定存在连接键.

为了有效解决已有链式多表连接算法中存在的上述问题，本文提出了 PipelineJoin 算法，它的主要思想是并行执行两条 MapReduce 任务流水线，并采用调度器对流水线中的多个 MapReduce 任务进行有序、高效的调度，以减少待执行 MapReduce 任务的等待时间，实现连接效率的最大化. 例如，对 n 个表 T_1、T_2、T_3、…、T_n 的链式连接，PipelineJoin 算法首先并行运行 2 个 MapReduce 任务，分别执行表 T_1、T_2 和表 T_3、T_4 的连接操作，生成连接结果集 S_1 和 S_2；然后，并行运行结果集 S_1、S_2 的连接和表 T_5、T_6 的连接，以此类推，以实现多表链式连接过程的流水线式执行. 它的关系代数式可以表示如下：

$$((\cdots(((T_1\infty T_2)\infty(T_3\infty T_4))\infty(T_5\infty T_6))\cdots)(T_{n-1}\infty T_n)) \tag{4}$$

相对于级联连接而言，PipelineJoin 大大减少了所需的 MapReduce 任务数量；相对于多路连接而言，PipelineJoin 消除了内存泄露的危险，可以实现大量表的连接；相对于 SmartJoin 算法而言，PipelineJoin 消除了对表的限制，不再局限于两个大表和多个小表，可以实现对任意规模表的连接操作，并且不需要两个大表之间必须存在连接键.

总的来说,本文的主要贡献如下:

(Ⅰ)提出了一种新的PlineJoin算法,它消除了现有算法对参与连接的表的诸多限制,可以实现任意数量、任意大小的多个表的链式连接操作;

(Ⅱ)算法采用了流水线调度模型,可以实现多个MapReduce任务的并行执行,大幅提升了多表的连接效率;

(Ⅲ)对不同规模、不同数量的表进行实验,结果表明PipelineJoin算法在处理多个大表的连接时,可以取得比已有算法更好的性能.

1 相关工作

MapReduce作为当前最流行的一个分布式并行计算框架,可以很好地解决大型数据集的计算问题,其中一个典型应用就是用MapReduce实现大型表的连接操作[1-9].

MapReduce没有提供对表的连接功能,要用MapReduce实现表的连接一般有3种方法:传统的MapReduce连接、基于改进的MapReduce连接和基于索引的MapReduce连接.传统的通过MapReduce实现连接的方法是通过自定义Map函数和Reduce函数实现连接功能,有时还需实现Combine函数和Merge函数,代表算法包括重分区连接、复制连接、半连接、分片半连接等[2].基于改进的MapReduce连接[2,8]是通过修改MapReduce框架来实现连接操作.文献[8]把MapReduce原来的两阶段Map-Reduce执行,增加为三阶段Map-Reduce-Merge执行,增加了一个Merge阶段,专门用来处理数据的连接操作.文献[2]在Map函数处理完成后,增加一个Join函数处理过程,然后再传输到Reduce函数进行处理.其中Join函数是从多个不同的表中读取数据,以实现多表连接.基于索引的MapReduce连接[6-7]利用索引的方式对表进行部分处理,以优化连接性能.文献[6]通过修改Hadoop的自定义函数,实现表间的连接,并利用寄宿索引的方法提高连接性能.文献[7]为待连接表增加一个标识其副本位置的Locator字段,通过改变Hadoop中副本放置策略优化连接,但是需要提前知道各表的位置信息.

已有的基于MapReduce的连接的大部分研究都是针对两表的[6-9],虽然多表连接可以用多个两表连接级联实现,但是这需要运行多个MapReduce任务,效率不高,所以专门针对多表连接的研究[1-5,10]应运而生.文献[10]处理的就是对多表进行按相似度级联连接的问题.文献[3]提出的多路连接算法虽然可以只采用一个MapReduce就实现多个表的连接,但是随着表数量的增加,运用这种算法产生的中间传输数据量将急剧增加,所以该算法只适合用于星型连接和表数不太多的链式连接,对于大量表的链式连接不适用.文献[4]中的算法思想与多路连接算法类似,但是它的应用更局限,只适用于三个表的连接.文献[5]采用2个MapReduce任务实现了非等值连接,一个MapReduce任务处理非等值连接,一个MapReduce任务处理多路连接.文献[2]虽然可以实现多表连接,但是它需要修改MapReduce框架.文献[1]提出的SmartJoin算法对表有太多限制.首先,它要求待连接的表需要是两个大表和多个存放在一个Reduce机器下的小表.其次,它还需要这两个大表之间必须存在连接键,对于星型连接而言,这个条件是可以满足的,但对于链式多表连接来说,则不能保证两个大表之间一定存在连接键.

PipelineJoin算法采用哈希连接[11]和优化重分区连接[2]的混合连接.文献[11]中的研究表明,哈希连接可以在多核处理器上表现出较好的性能.此外,本文还创建了一个调度器用于实现对MapReduce任务的流水化调度,文献[12-13]也对调度器进行了相关研究.文献[13]采用ZooKeeper对MapReduce任务进行协调调度,文献[12]中也提出了3个对任务处理器进行调度的近似算法.

2 背景

2.1 MapReduce

MapReduce是当前比较流行的一个用于处理数据密集型并行计算的框架.如图1所示,在MapReduce中,输入数据集根据集群中节点进行划分,并以<键,值>对的形式存储在分布式文件系统上(如GFS和HDFS).MapReduce采用两个函数(Map函数和Reduce函数)来实现并行计算,Map函数把输入的键值对转换成键值列表的形式,Reduce函数再对该键值列表进行规约生成较小的键值对集合,并将结果保存到分布式文件系统上.

2.2 基于MapReduce的连接算法

现有的基于MapReduce的连接算法主要包括:优化重分区连接、复制连接、半连接和分片半连接等[2].本文的PipelineJoin算法是优化重分区连接和哈希连接的混合,在处理输入表的连接时采用优

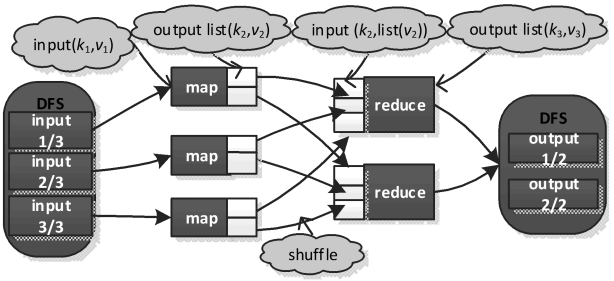

图 1　MapReduce 过程示意图

Fig. 1　The course of MapReduce

化重分区连接；然后对重分区后的连接结果进行处理；最后在本地机器上运行哈希连接，以完成多个大型表的连接操作. 在介绍 PipelineJoin 算法前，下面先介绍一下优化重分区连接和哈希连接.

2.2.1　优化重分区连接

优化重分区连接利用 MapReduce 的“排序-合并”机制来分组数据，使用一个单独的 MapReduce 任务来实现连接. 例如，采用优化重分区连接执行表 A 和 B（假设 B 是小表）的 MapReduce 过程，如图 2 所示.

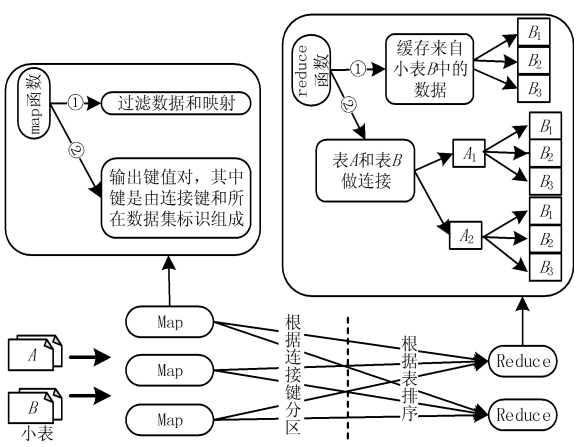

图 2　优化重分区连接示意图

Fig. 2　Optimized re-partition connection

在 Map 阶段，Map 函数从 HDFS 中读入待连接的两个表 A 和 B，并以键值对的形式输出需要连接的数据. 其中，键是这两个表的连接键和一个标记数据属于哪个表的标识；值是连接结果中需要显示的内容和一个标记数据属于哪个表的标识. 对键值对添加完标识后，Map 机器通过自定义分区函数对这些键值对进行分区. 默认分区函数是以键进行分区的. 这里根据连接键进行分区，以保证相同连接键的键值对可以被发送到同一个 Reduce 机器.

在 Reduce 阶段，一个 Reduce 机器接收从 Map 机器传来的某个输出键的所有输出值，这些输出值可能来自不同 Map 机器，所以 Reduce 机器需要对这些键值对进行排序. 这里根据键中的标识进行排序，以保证小表中的数据先于大表中数据被 Reduce 函数接收. 输入数据从 HDFS 数据流中读取，读取完成后，Reduce 函数按照不同来源的表对输入数据分别进行处理，当输入数据来自小表时，需要缓存到 Reduce 机器上. 当输入数据来自大表时，直接与 Reduce 机器缓存中的数据做笛卡尔积连接操作；最后，把连接后的结果存储在 HDFS 中，完成优化重分区连接.

2.2.2　哈希连接

哈希连接是两表连接算法中最传统的算法，包括两个阶段：构建阶段和探测阶段. 在构建阶段，小表被加载到内存的哈希表中；在探测阶段，扫描分析大表中的数据，如果在哈希表中存在相同连接值的数据则进行连接操作；否则，顺序扫描下一条数据，直至完成连接.

3　PipelineJoin 算法

PipelineJoin 算法的主要思想是采用流水线模型和调度器来实现 MapReduce 任务的流水线式执行，从而有效提高多表连接的效率. 同时，PipelineJoin 可以有效克服已有链式多表连接算法的缺陷.

3.1　流水线模型

对于待连接的 n 个表 T_1、T_2、T_3、…、T_n，首先需要把这 n 个表进行两两分组，相邻两个表分为同一组（对于链式连接，相邻两个表间存在连接键），如 T_1、T_2 一组，T_3、T_4 一组；然后并行运行两个 MapReduce 任务依次对每组表进行处理. 对于奇数个表的情况，在最后增加一个空表，组成偶数个表.

在一个集群中并行执行两个 MapReduce 任务时，可以共用 Map 机器，这些 Map 机器称为一个“Map 机器群”. 这里把 Reduce 机器等分为两份，一份称为“Reduce_1 机器群”，另一份称为“Reduce_2 机器群”. 对于这 n 个表执行 MapReduce 连接的任务调度过程如图 3 所示. 其中，$(i=1,3,\cdots,n-1)$ 表示表 T_i 和 T_{i+1} 执行优化重分区连接的 Map 任务，$(i=1,3,\cdots,n-1)$ 表示表 T_i 和 T_{i+1} 执行优化重分区连接的 Reduce 任务，$(j=1,2,\cdots,(n/2)-1)$ 表示连接结果集间执行的第 j 次哈希连接.

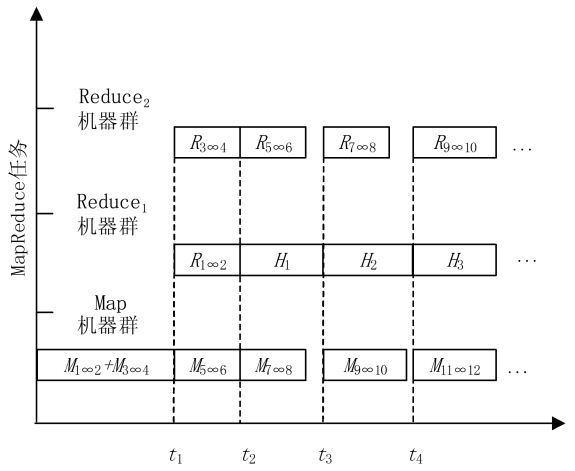

图3 MapReduce任务执行流水线图

Fig. 3 Task execution pipeline of MapReduce

采用流水线模型执行MapRecuce任务时，首先从HDFS上读取前4个表T_1、T_2、T_3、T_4；然后，在Map机器群上，对这4个表执行Map操作，并把Map操作的输出数据传送给相应的Reduce机器群执行. 例如，由T_1、T_2表得到的Map输出数据传送给Reduce_1机器群，由T_3、T_4表得到的Map输出数据传送给Reduce_2机器群. 在T_1时刻，Map机器群上的任务完成，并把处理后的数据传输给对应Reduce机器群进行优化重分区连接，同时在Map机器群上运行表T_5和T_6的Map任务. 在T_2时刻，机器群完成表T_1、T_2和T_3、T_4的优化重分区连接，此后需要再对T_1、T_2和T_3、T_4的优化重分区连接后的结果集在Reduce_1机器群上进行哈希连接，T_2时刻还需要在Map机器群上运行表T_7和T_8的Map任务，在Reduce_2机器群上运行表T_5和T_6的优化重分区连接. 在T_3时刻，Reduce_1机器群上的哈希连接H_1结束，此时，需要再次在Map机器群上运行表T_9和T_{10}的Map任务，在Reduce_2机器群上运行表T_7和T_8的优化重分区连接，在Reduce_1机器群上进行H_1连接后结果集和表T_5、T_6的连接结果集的哈希连接H_2. 以此类推，这样一直流水线式运行下去，直到完成所有表的连接.

3.2 调度器

本文通过引入一个调度器来控制流水线上各个操作执行的时间点，该调度器控制着集群中机器的任务分配，并维护着集群中机器的状态信息. 调度器维护着一个记录集群中所有机器信息的表，该表记录着机器的位置信息、机器是执行Map任务还是执行Reduce任务、是位于Reduce_1机器群还是位于Reduce_2机器群以及机器是否忙碌等信息. Map机器群执行的是所有表的Map操作，当Map机器群执行完一组表的Map操作后，调度器按顺序把下一组表发送给它继续执行. Reduce_1机器群执行的是T_1和T_2表的优化重分区连接的Reduce任务和所有结果集的哈希连接，当T_1、T_2的优化重分区连接完成后，它向调度器询问Reduce_2机器群上T_3、T_4表是否也已完成连接，若完成，Reduce_1机器群接着执行T_1、T_2表的连接结果集和T_3、T_4表的连接结果集之间的哈希连接，否则进入等待状态，并持续向调度器发起询问，直到Reduce_2机器群完成连接. Reduce_2机器群执行的是除T_1和T_2表外其他所有表的优化重分区连接. 它在执行完成T_3、T_4表的优化重分区连接后，向调度器询问Reduce_1机器群上任务是否已经完成，若完成，它会继续执行T_5、T_6表的优化重分区连接，否则进入等待状态，并不断向调度器发起询问，直到Reduce_1机器群完成连接.

3.3 算法设计

PipelineJoin算法的具体执行过程如算法1所示，如果输入奇数个表，则需向待连接表中添加一个空表(第1～4行)；然后把输入表T_1、T_2、T_3、T_4分别分配给Map机器群进行Map操作，并把Map结果发给对应的Reduce机器(第5行). 同时在Reduce_1机器群和Reduce_2机器群上执行表T_1、T_2和T_3、T_4的优化重分区连接，分别生成结果集S_1(为了方便下面描述这里记作H_0)、S_2，S_1和S_2需要按下一个连接键进行排序. 这里下一个连接键指的是该结果集下一次执行连接时的连接键，在Map机器群执行表T_5、T_6的Map操作(第6～7行)；然后，对Reduce_1机器群中结果集S_1在Reduce_1机器群内部按键传输，Reduce_2机器群中结果集按键值进行逻辑划分(第8行)；接着，把S_2中逻辑划分后的数据，分别传送给Reduce_1机器群中对应机器(第9行). 然后，并行执行以下操作：①在Reduce_1机器群上执行H_{j-1}、S_{j+1}的哈希连接，生成结果集H_j，并对H_j按下一个连接键进行排序；②在Reduce_2机器群上执行表T_i、T_{i+1}的优化重分区连接，并把结果集S_{j+2}进行逻辑划分后传输到Reduce_1机器群中对应机器上；③在Map机器群上，执行表T_{i+2}、T_{i+3}的Map操作(第9～12步). 循环执行第11步，直至完成所有表的连接；最后输出连接结果，算法结束.

算法3.1 PipelineJoin输入：

待连接表 $T_1, T_2, T_3, \cdots, T_n$

输出：$T_1 \infty T_2 \infty T_3 \infty \cdots \infty T_n$ 的连接结果

begin

1. if n is an odd number

2. add a null table T_{n+1};

3. $n=n+1$;

4. end if

5. execute map(T_1), map(T_2), map(T_3) and map(T_4) in parallel;

//并行执行表 T_1, T_2, T_3, T_4 的 Map 操作

6. execute

$H_0=S_1=$OptimaizedRepartitionjoin(T_1, T_2),

//T_1、T_2 优化重分区连接,生成结果集 $S_1(H_0)$

$S_2=$OptimaizedRepartitionjoin(T_3, T_4),

//T_3、T_4 优化重分区连接,生成结果集 S_2

map(T_5), map(T_6) in parallel;

//T_5、T_6 的 Map 操作

7. execute Sort(S_1), Sort(S_2) in parallel;

//并行对连接结果集 S_1、S_2 排序

8. execute InTranslate(S_1),LPatition(S_2)

in parallel;

//对 S_1 中数据按键传输,S_2 中数据逻辑划分

9. OutTranslate(S_2);

//S_2 中数据传到 Reduce_2 机器群中对应机器

10. for $j=1$ to $(n/2)-1$, $i=2j+3$ do

11. $H_j=$HashJoin(H_{j-1}, S_{j+1})and Sort(H_j),

//(H_{j-1}、S_{j+1}的哈希连接并对结果集 H_j 排序

$S_{j+2}=$OptimaizedRepartitionjoin(T_i, T_{i+1}),

//T_i、T_{i+1}的优化重分区连接

LPatition(S_{j+2}), OutTranslate(S_{j+2}),

//S_{j+2}的逻辑划分与传输

map(T_{i+2}), map(T_{i+3}) in parallel;

//T_{i+2}、T_{i+3}的 Map 操作

13. end for

end

3.4　算法实例分析

下面以 $T_1(a,b)$、$T_2(b,c)$、$T_3(c,d)$、$T_4(d,e)$、$T_5(e,f)$、$T_6(f,g)$这 6 个表的链式连接为例,介绍 Reduce 机器群的工作过程.

首先在 Reduce_1 机器群和 Reduce_2 机器群上并行执行表 T_1、T_2 和表 T_3、T_4 的优化重分区连接,生成结果集 S_1、S_2;然后对 S_1、S_2 按下一个连接键进行排序,因为下一次 S_1 要与 S_2 进行连接,所以它的下一个连接键是 c;同理,S_2 的下一个连接键也是 c.接着,并行执行 S_1、S_2 的哈希连接(生成结果集 H_1)和 T_5、T_6 的优化重分区连接(生成结果集 S_3);最后执行 H_1 与 S_3 的哈希连接,生成 T_1、T_2、T_3、T_4、T_5、T_6 这 6 个表的连接结果.

两表的优化重分区连接在 2.2.1 节已进行详细描述,不再赘述,这里我们对结果集 S_1、S_2 间的哈希连接进行描述,其他结果集间的哈希连接与此类似.我们将对传统的哈希连接和本文改进后的哈希连接进行比较.

(Ⅰ)传统的哈希连接.一般而言,要完成处在不同机器上结果集 S_1、S_2 之间的哈希连接,需要把一个结果集的全部数据发送到所有含有另一个结果集的机器上.我们把 S_2 结果集发送到 Reduce_1 机器群上做连接.这样做的缺陷是,一方面,数据的传输量很大,Reduce_1 机器群上有多少机器,就需传输多少份 S_2 结果集;另一方面,Reduce_1 机器群中的机器在完成优化重分区连接后可能生成相同的连接结果,这些相同的连接结果,在下次连接时会进行重复连接.

图 4 描述了采用传统连接方法,Reduce_1 机器群上哈希连接的情况.假设 Reduce_1 机器群中机器 1 执行的是表 T_1、T_2 连接键的值为 1 的键值对(记为 T_{11}、T_{21})的连接,生成重分区连接结果集记为 S_{11},哈希连接结果集记为 H_{11}.Reduce_1 机器群中机器 2 执行的是表 T_1、T_2 连接键的值为 5 的键值对(记为 T_{12}、T_{22})的连接,生成重分区连接结果集记为 S_{12},哈希连接结果集记为 H_{12}.以此类推,Reduce_1 机器群中机器 n 上执行连接后生成的重分区连接结果集记为 S_{1n},且 S_{11}、S_{12}、…、S_{1n}共同组成 S_1,哈希连接结果集记为 H_{1n},且 H_{11}、H_{12}、…、H_{1n} 共同组成 H_1.Reduce_2 机器群中机器 1 执行的是表 T_3、T_4 连接键的值为 1 的键值对(记为 T_{31}、T_{41})的连接,生成结果集记为 S_{21}.Reduce_2 机器群中机器 n 上执行连接后生成的结果集记为 S_{2n},且 S_{21}、S_{22}、…、S_{2n}共同组成 S_2.

图 4 中,T_1、T_2 经过优化重分区连接后,结果集 S_{11}、S_{12}上均存在连接结果(1,3).当它们接收到来自 Reduce_2 机器群的 S_2 进行连接时,会产生很多相同的连接结果,如(2,3)、(5,3)等,S_2 中有多少连接键值为 1 的键值对,结果集 H_{11}、H_{12}上就会有多少重复的连接结果.

(Ⅱ)本文改进后的哈希连接针对采用传统方法进行哈希连接时存在的不足,对 Reduce_1 机器群中优化重分区后的结果集按键的值分别进行传送,从而有效地避免了重复连接的情况.

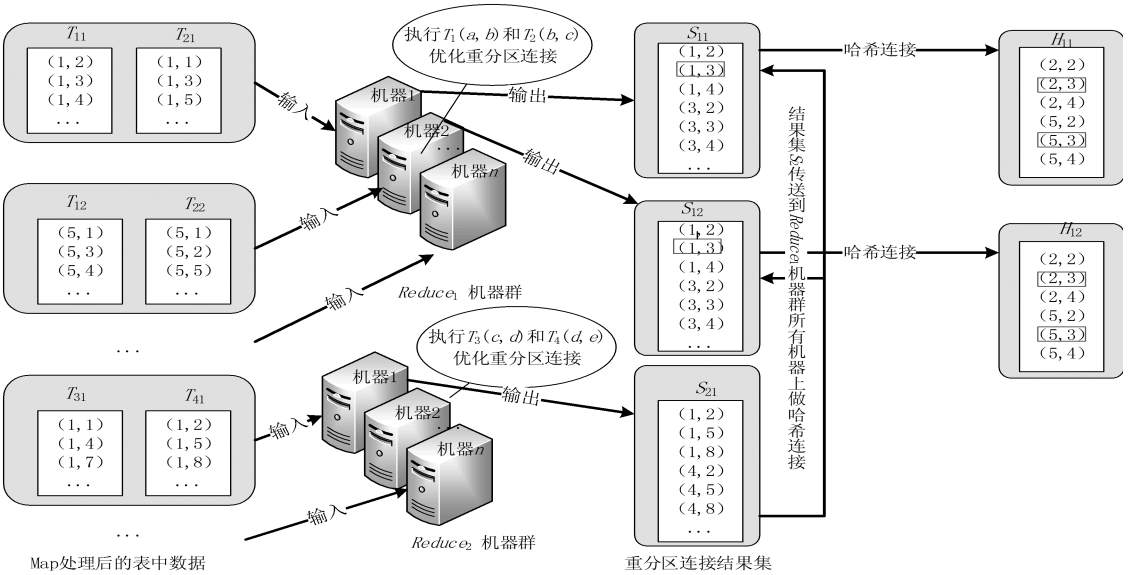

图 4 采用传统方法时 Reduce 机器群执行连接的过程

Fig. 4 The process of performing a connection to Reduce machine group by traditional method

采用优化方法后，一方面，结果集 S_2 只需向 $Reduce_1$ 机器群上传送一份（把相应数据发送给对应机器即可）；另一方面，$Reduce_1$ 机器群的机器上不会出现重复的连接结果，因为拥有相同连接键的连接结果存储在同一台机器上，这样可以保证每台机器数据不重复. 实现该优化方法时，需要调度器中记录着所有 $Reduce_1$ 机器群中机器可以接收的键的值，以保证 $Reduce_2$ 机器群上机器直接把连接键相同的数据发送给 $Reduce_1$ 机器群上的对应机器.

图 5 给出了采用改进后的优化方法在 Reduce 机器群执行连接的过程. 由图 5 可知，T_1、T_2 经过优化重分区连接后，结果集 S_{11}、S_{12} 上含有相同连接结果(1,3). 改进后的方法需要按键值在 $Reduce_1$ 机器群中进行传输，结果集 S_1 中所有连接结果(1,3)均会传输到机器 1 上，这样就使得不同机器上不会存在相同的连接结果，也就避免在下一次连接中进行重复的连接.

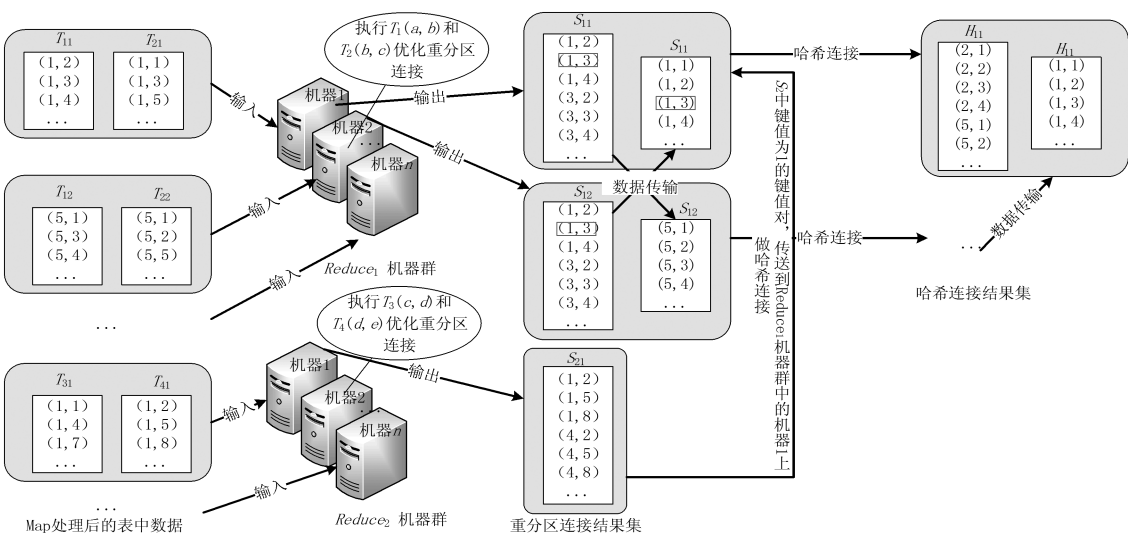

图 5 采用优化方法时 Reduce 机器群执行连接的过程

Fig. 5 The process of performing a connection to Reduce machine group by optimized method

3.5　性能分析

与已有的多表连接算法相比，PipelineJoin 算法具有更好的性能.

相对于级联连接而言，PipelineJoin 大幅减少了连接过程所需的 MapReduce 任务的数量. 级联连接采用迭代的连接方法依次执行多个两表连接，而 PipelineJoin 算法采用流水线模型，并行运行 2 条 MapReduce 任务流，所以对于单条 MapReduce 任务流而言，PipelineJoin 所需的 MapReduce 任务数量只是级联连接的一半.

相对于多路连接而言，PipelineJoin 消除了内存泄露的危险，可以实现大量表之间的连接. 多路连接采用哈希桶的方式，只运行一个 MapReduce 任务就可完成整个多表连接，但随着参与连接的表的数量的增加，需要传送给同一台 Reduce 机器的数据量呈线性增加，最终会占满内存，造成内存泄露. PipelineJoin 采用多个 MapReduce 任务，不需要把所有数据都存放在同一台 Reduce 机器上，因此不会发生内存泄露问题.

相对于 SmartJoin 算法而言，PipelineJoin 消除了对表的限制，不再局限于两个大表和多个小表，可以实现对任意规模表的连接操作，并且不需要两个大表之间必须存在连接键. 运用 SmartJoin 算法时，需要把多个小表同时存储在同一个 Reduce 机器上，这样完成两个大表的连接后可以直接与本机的小表连接，这使得它的应用场景受到很大限制. 相反，PipelineJoin 则可以处理多个大表的连接，不存在对表大小的限制.

通过上述比较可以发现，PipelineJoin 可以：克服多路连接算法和 SmartJoin 算法存在的缺陷，有效解决多个大表之间的连接；在处理大表间的连接时，比级联连接减少了需要执行的 MapReduce 任务数，有效缩短了连接所需时间.

4　实验设计与结果分析

4.1　实验环境

本文在 Hadoop 环境下实现了 PipelineJoin 算法，并进行了全面测试，以验证 PipelineJoin 算法的良好性能. 由前面 3.5 节对各种已有多表连接算法的性能分析可知，多路连接算法和 SmartJoin 算法并不适于处理多个大表之间的链式连接，所以本文只对 PipelineJoin 算法与级联连接进行实验对比分析，以验证二者的性能差异.

本文在阿里云平台上搭建了实验环境，使用了 9 个节点的集群. 其中，有 1 个节点配置为 NameNode 和 JobTracker，负责调度任务，其余 8 个节点配置为 DataNode 和 TaskTracker，同时负责存储和计算. 每个节点的硬件配置为双核 2.3 GHz CPU，4 GB 内存，60 GB 机械硬盘. 软件系统为 64 位 CentOS 6.5，使用的 Hadoop 版本为原生 Hadoop 2.6.0，Java 环境为 OpenJDK7.

4.2　实验数据

实验所使用的数据集采用 TPC-H（http://www.tpc.org/tpch/）生成. 为了比较不同算法在连接不同数量表时的性能差异，我们生成了多个表，并将表编号为 T_1、T_2、T_3、…、T_n. 其中，每个表均包含 2 000 000 条记录. 同时，我们对数据做了一定修改，使每条记录的长度一致，并使相邻两个表中均包含一个相同的字段以进行连接.

通常，两个表进行连接后，输出的连接结果的记录数量取决于连接键值的分布情况，这一记录数量有可能与参与连接的表的记录数量相当，也有可能增多或减少. 对于级联算法而言，首先需要进行两个表的连接，所产生的中间结果再与其他表依次进行连接，因此中间结果的大小会对算法的执行时间产生影响.

为了充分模拟各类应用场景，本实验分别构建了 3 种连接键值分布情况，记为 A、B、C. 对于键值分布 A，随着参与连接的表个数的增加，中间结果和最终结果的记录数量逐渐减少. 相应地，键值分布 B 则保持不变，而键值分布 C 则逐渐增加.

假设参与连接的每个表的记录数量均相等，则在每种键值分布情况下，输出结果的记录数量如图 6 所示. 其中，横坐标表示参与连接的表的个数，纵坐标则表示输出最终输出结果的记录数量.

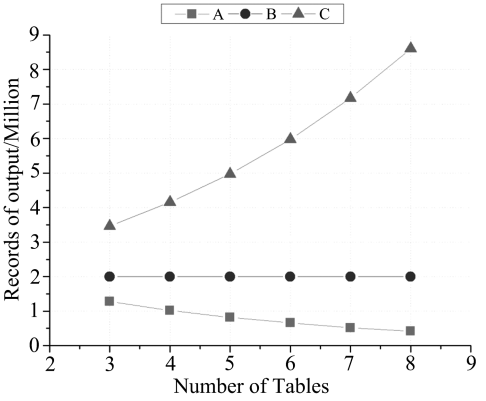

图 6　输出结果的数量与键值分布和表个数的关系

Fig. 6　The relations between numbers of output results, key values distribution and table numbers

4.3 网络 I/O

本文比较 PipelineJoin 算法与级联连接算法为完成连接所需要的网络 I/O 代价. Hadoop 的准则之一是“移动计算比移动数据更经济”,即尽量减少数据在网络上的传输. 随着节点数量的增加,集群可以获得更加强大的计算能力,但节点的增加会平摊集群间的网络带宽,导致数据在网络间的传输变慢,因此减少网络 I/O 代价能可提升连接操作的性能.

传统的级联连接算法会产生大量的网络 I/O 代价,每次连接两个表时,都需要将生成的中间结果写入到分布式文件系统 HDFS 中,在执行下一个连接时再进行读取,因此需要多次读取数据.

在不同的连接键分布情况下,PipelineJoin 算法和级联连接算法所产生的网络 I/O 代价如图 7 所示. 其中,横坐标表示参与连接的表个数,纵坐标对应不同算法分别需要读取、传输、写入的记录数量. 这里的“读取”指的是 Map 任务从 HDFS 中读取数据,“传输”指的是将数据从 Map 端发送至 Reduce 端,“写入”指的是将结果写入到 HDFS 中.

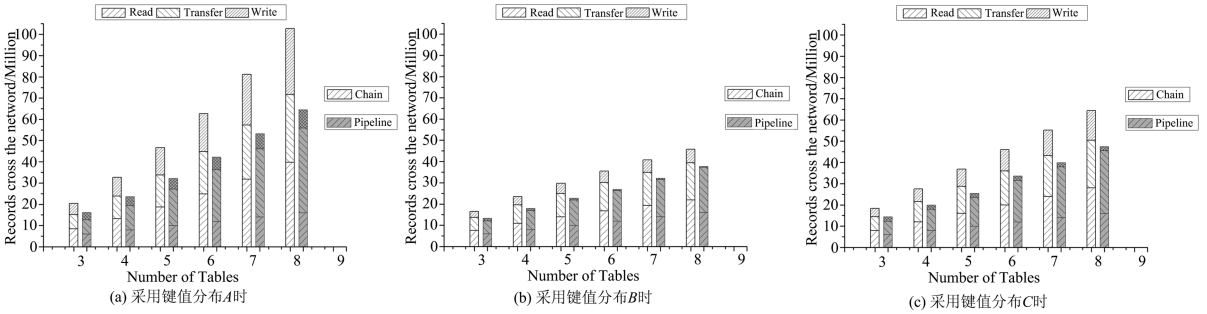

图 7 采用不同键值分布时在网络中传输的记录数量

Fig. 7 Records in the network transmission by different key values distribution

从图 7 可以看出,随着参与连接的表个数的增加,级联连接算法的网络 I/O 增加明显,特别是在键值分布 C 的情况下,由于每次连接两个表所产生的中间结果记录数量逐渐增加,导致读取和写入产生了大量网络 I/O. 采用 PipelineJoin 算法时,所有的表都只需要读取一次,大幅降低了网络 I/O,并且 PipelineJoin 只在任务最后才写入结果,写入 HDFS 所产生的网络 I/O 远远小于级联连接算法. 在 Hadoop 中,Map 任务读取的表通常是存储在当前节点的表,因此这部分数据无需通过网络传输,但是在写入结果时,需要写入到多个副本中,这需要通过网络进行数据传输. 因此,PipelineJoin 算法相比于级联连接算法而言,在传输和写入时能节约大量时间.

4.4 响应时间

通过网络 I/O 代价的对比可以看出,PipelineJoin 算法可以减少网络 I/O 代价进而提升性能.

图 8 是在不同的连接键分布情况下,PipelineJoin 算法和级联连接算法完成整个连接任务的响应时间. 从图 8 可以看出,PipelineJoin 算法在三种键值分布情况下,所需的连接时间均小于级联算法,且随着连接表个数的增加,PipelineJoin 算法的优势更加明显. 这是因为随着表个数的增加,级联算法需要写入到 HDFS 中的结果数量随之增加,显著降低了整个连接任务的性能. PipelineJoin 算法只需要在任务的最后才将结果写入到 HDFS 中,减少了网络开销,提升了整体性能.

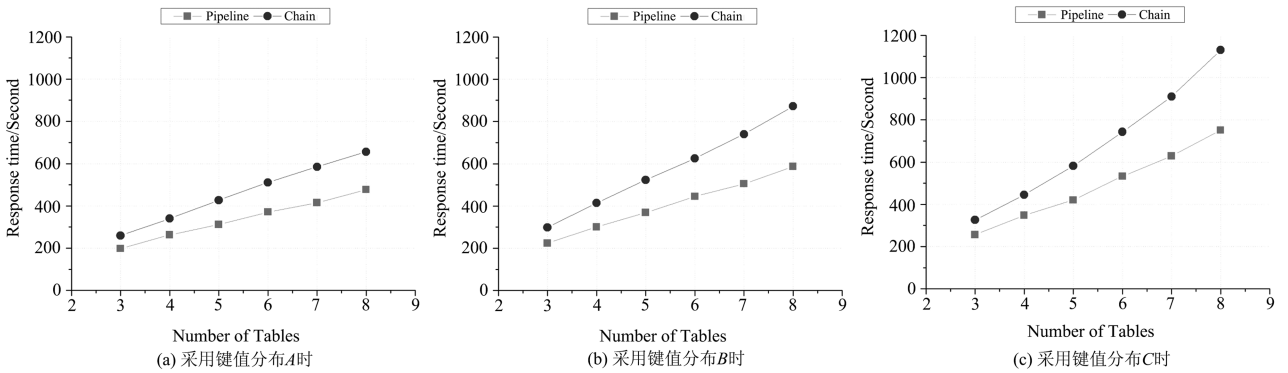

图 8 采用不同键值分布时连接任务的响应时间

Fig. 8 The response time of the connection task by different key values distribution

6　结论

基于 MapReduce 的多表连接算法已经存在很多研究，但是，它们或者无法处理大型表的连接，或者由于需要顺序执行较多的 MapReduce 任务而导致效率较低. 本文提出了一个基于 MapReduce 的多表连接算法—— PipelineJoin，它可以高效地实现任意多个表的链式连接. PipelineJoin 通过引入一个调度器来控制 MapReduce 任务的流水线式执行，以实现相同时间内 MapReduce 任务吞吐量的最大化. 与已有链式多表连接算法相比，PipelineJoin 算法可以更高效地处理多个大型表的连接操作. 另外，本文还给出了任务调度流水线模型，形象地描述了任务的调度过程. 最后，大量实验表明，本文算法与原有链式多表连接算法相比，可以有效减少连接时间，并且可以克服已有算法的缺陷. 在未来的研究工作中，我们将把本文算法的核心思想进一步应用到星型连接和链式连接的混合连接中，有效提升混合连接的效率.

参考文献(References)

[1] Slagter K, Hsu C H, Chung Y C, et al. SmartJoin: A network-aware multiway join for MapReduce [J]. Cluster Computing, 2014, 17(3): 629-641.

[2] Jiang D W, Tung A K H, Chen G. MAP-JOIN-REDUCE: toward scalable and efficient data analysis on large clusters[J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(9): 1299-1311.

[3] Afrati F N, Ullman J D. Optimizing multiway joins in a map-reduce environment [J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(9): 1282-1298.

[4] Kimmett, Thomo A, Venkatesh S. Three-way joins on MapReduce: An experimental study[C]// Proceedings of the 5th International Conference on Information, Intelligence, Systems and Applications. Crete, Greece: IEEE Press, 2014: 227-232.

[5] Yan K, Zhu H. Two MRJs for multi-way theta-join in MapReduce[C]// Proceedings of the 6th International Conference on Internet and Distributed Computing Systems. Hangzhou, China: Springer, 2013: 321-332.

[6] Dittrich J, Quiané-Ruiz J A, Jindal A, et al. Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing) [C]// Proceedings of the 36th International Conference on Very Large Data Bases. Singapore: ACM Press, 2010: 518-529.

[7] Eltabakh M Y, Tian Y Y, ? zcan F, et al. Cohadoop: Flexible data placement and its exploitation in hadoop [C]// Proceedings of the 37th International Conference on Very Large Data Bases. Seattle, USA: ACM Press, 2011: 575-585.

[8] Yang H C, Dasdan A, Hsiao R L, et al. Map-reduce-merge: Simplified relational data processing on large clusters [C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. Beijing, China: ACM Press, 2007: 1029-1040.

[9] Blanas S, Patel J M, Ercegovac V, et al. A comparison of join algorithms for log processing in MapReduce [C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. Indianapolis, USA: ACM Press, 2010: 975-986.

[10] Zhang X X, Guo Z L, Guo H L, et al. CasJoin: A Cascade Chain for Text Similarity Joins [C]// Proceedings of the 19th ACM international conference on Information and knowledge management. New York, USA: ACM Press, 2010: 1725-1728.

[11] Blanas S, Li Y N, Patel J M. Design and evaluation of main memory hash join algorithms for multi-core CPUs [C]// Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. Athens, Greece: ACM Press, 2011: 37-48.

[12] Yuan Y, Wang D, Liu J C. Joint scheduling of MapReduce jobs with servers: Performance bounds and experiments[C]// Proceedings of the IEEE Conference on Computer Communications. Toronto, Canada: ACM Press, 2014: 2175-2183.

[13] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: Wait-free coordination for Internet-scale systems[C]// Proceedings of the Annual Technical Conference. Boston, USA: ACM Press, 2010: 11-24.