



CC-MRSJ:Hadoop 平台下缓存敏感的星型联接算法^{*}

周国亮^{1,2},朱永利¹,王桂兰¹

(1.华北电力大学控制与计算机工程学院 保定 071003;

2.国网冀北电力有限公司技能培训中心 保定 071051)

摘要:提出了一种缓存敏感的 MapReduce 星型联接算法,事实表每列单独存储,维表根据维层次划分为多个列簇。事实表外键列与对应维表采用相关性存储,减少联接过程中的数据移动。算法分为两个阶段,首先每个外键列和对应维表进行联接;然后对中间结果进行联接,随机访问测度列,进而得到最终结果。算法只读取需要的数据,缓存利用率高,从而具有良好的缓存敏感特性;算法充分利用时延实体化,避免不必要的数据库访问和移动。通过在 SSB 数据集上与 Hive 系统的对比测试表明,CC-MRSJ 算法具有较高的执行效率。

关键词:星型联接;MapReduce;缓存敏感;存储模型

doi: 10.3969/j.issn.1000-0801.2013.10.007

CC-MRSJ: Cache Conscious Star Join Algorithm on Hadoop Platform

Zhou Guoliang^{1,2}, Zhu Yongli¹, Wang Guilan¹

(1. College of Control and Computer Engineering, North China Electric Power University, Baoding 071003, China;

2. Skill Training Center, State Grid Jibei Electric Power Company Limited, Baoding 071051, China)

Abstract: A cache-conscious MapReduce star join algorithm was presented, each column of fact table was separately stored, and dimension table was divided into several column families according to dimension hierarchy. Fact table foreign key column and corresponding dimension table was co-location storage, thus reducing data movement in the join process. CC-MRSJ consists of two phases: firstly each foreign key column and the corresponding dimension table were joined; then the intermediate results were joined and random accessed measure columns, and so got the final result. CC-MRSJ read only the data needed, and cache utilization is high, so it has good cache conscious feature; it also takes advantage of late materialization, avoiding unnecessary data access and movement. CC-MRSJ has higher performance comparing to hive system based on SSB datasets.

Key words: star join, MapReduce, cache conscious, storage model

1 引言

随着大数据在各个行业的迅速发展,传统的并行数据仓库系统在处理大数据时由于成本高、可扩展性差及容错能力低等原因而力不从心。因此,基于 MapReduce^[1,2]技术

的数据仓库系统获得了广泛关注和应用,如 Facebook 的 Hive^[3]、Yahoo 的 Pig^[4]等。然而,困扰 MapReduce 数据仓库系统进一步深入应用的主要问题是性能低下。因此,围绕如何提高 MapReduce 系统的性能是目前各方普遍关注的热点难点。在 MapReduce 环境下,数据一般以分布式文件系

^{*} 中央高校基本科研业务费专项基金资助项目(No.13MS103),河北省高等学校科学研究基金资助项目(No.Z2011306)



统的形式存储在集群的磁盘上,其主要代价是磁盘和网络 I/O,而不同的数据存储格式对系统的 I/O 有较大的影响,从而影响系统性能。在传统数据仓库系统中普遍采用列存储提高性能^[5-7]。通过列存储可以有效地减少 I/O,并利用数据压缩技术,进一步提高性能。因此,随着 MapReduce 技术的流行,列存储在 MapReduce 系统中获得广泛关注,提出了 RCFile^[8]、COF^[9]和 CFile^[10]等列存储文件格式。基于列存储的数据结构,MapReduce 性能获得了一定提高。

在数据仓库中,通常采用星型模式作为数据组织和存储模型。一般情况下,星型模型由一个很大的事实表和多个较小的维表组成。因此在数据仓库中,星型联接(star join)是一项核心和基本的操作,即事实表和多个维表之间的联接。MapReduce 中已有的联接算法^[10-12]较少考虑星型联接的特殊性,而造成事实表数据移动和庞大的中间结果,因此性能很难提高。考虑星型模型的结构,提高性能的关键是尽量减少事实表的磁盘 I/O 和数据移动,事实表进行本地处理。

另一方面,MapReduce 起初部署在配置较低的机器上,内存与磁盘的存取速度差距不大,较少考虑现代处理器的多层存储体系结构^[13],造成 CPU 缓存利用率低。而目前的 MapReduce 普遍部署在配置很高的机器上,内存与磁盘的读写速度差距越来越大,因此充分利用现代处理器的缓存特性^[14],有利于进一步提高 MapReduce 算法的性能^[15,16]。

本文提出了一种具有缓存敏感特性的 CC-MRSJ(cache-conscious MapReduce star join)算法,首先对事实表进行垂直划分,每列单独存储;维表根据具有的维层次结构划分为不同的列簇;外键列和对应维表采用相关性存储,尽量部署在相同的节点上。通过上述存储结构,可以提高缓存利用率,减少数据在节点间的移动。算法通过两个阶段完成,首先外键列和对应维表进行联接,可以采用 map-reduce 联接或 map 联接;然后对中间结果根据位置索引进行 reduce 联接并输出,利用位置索引随机读取查询中涉及的测度列。通过对 SSB^[17]数据集的 Q3.1 和 Q4.1 测试表明,CC-MRSJ 算法具有较高的执行效率。

2 相关工作

2.1 MapReduce 及其数据存储格式

MapReduce 是由 Google 公司近年来提出的一种大数

据处理框架,MapReduce 程序一般分为 map 和 reduce 两个阶段。map 函数接受键值对(key-value pair)输入,产生一组中间结果键值对,MapReduce 框架会将 map 函数生成结果中键相同的值传递给一个 reduce 函数;reduce 函数接受一个键以及相关的一组值,将这组值进行合并产生一组规模更小的值。

MapReduce 的思想是将计算推向数据,尽量减少数据移动。目前,由 Apache 基金会组织开发的 MapReduce 开源实现 Hadoop^[18]在工业界和学术界获得了广泛应用。

MapReduce 中的数据主要存储在磁盘,程序运行中的主要代价是磁盘和网络 I/O。因而不同的数据存储格式对性能具有较大的影响,通过列存储可以有效减少 I/O,进而提高性能,因此列存储技术在 MapReduce 中获得了广泛关注,多种列存储模型被提出。列存储中一般通过位置索引(隐式 RowId)组织不同列的数据。目前,Hive 系统提供 RCFile(record columnar file)^[8]格式数据存储,RCFile 首先将数据水平划分为数据块,然后在块内垂直划分为列簇。而 CFile^[10]不考虑水平划分,而是将表垂直划分为多个列簇。CIF^[9]采用每列单独存储,不考虑列簇,但不同列的数据采用相关性存储,从而减少列组合时的网络数据传输。然而这些存储模型并没有考虑星型模型的特殊结构。

2.2 联接算法

联接操作是数据库中的一项核心操作,研究人员对此进行了深入系统的研究。而随着 MapReduce 对多数据源分析需求的增加,研究人员提出了多种 MapReduce 环境下的联接算法,目前 Hive 系统中主要支持的联接算法如下。

(1) map-reduce 联接

在 map 阶段,两个表根据联接关键字进行划分和排序;在 reduce 阶段,对每一个分区进行联接操作。这种方式类似于关系数据库的散列联接(Hash join)。

(2) map 联接

假如参加联接操作的两个表中有一个足够小,小表被分发到各个节点,于是在 map 阶段大表的每个片段与小表进行联接。这种方式类似于数据库中的嵌套循环联接(nested loop join)。

(3) 分块的 map 联接

将参加联接的表划分为对应的数据块,于是在 map 阶段一个表中的块与另一表中对应的块进行联接,不需要 reduce 操作。这种方式类似于 Sort-Merge 联接中的 Merge 操作。

星型联接是数据仓库中非常普遍的一种操作,通常情

况下星型联接是一个事实表 F 和多个维表 $D_1, D_2, \dots, D_n$ 之间的联接。事实表和维表具有如下特征:

- 维表 D_i 的主键 PK 和事实表 F 的外键列 FK_i 之间存在对应关系;
- 事实表一般由外键列和测度列组成: $F(fk_1, fk_2, \dots, fk_n, m_1, m_2, \dots, m_k)$ 。

为了在 MapReduce 环境下高效处理星型联接, 相关研究人员提出了并发联接 (concurrent join)^[10] 和 Scatter-Gather-Merge (SGM)^[12] 算法。这两个算法的核心思想都是把星型联接划分为多个小的联接, 然后再合并中间结果。并发联接算法采用垂直划分, SGM 采用水平划分。但两者未考虑事实表和维表相关性存储及缓存特性。

3 CC-MRSJ 算法

3.1 数据存储格式

列存储一般将所有列划分为不同的列簇, 经常一起访问的列组成一个簇, 从而减少了列组合实体化 (materialization) 的代价, 也避免了访问不相关的列。但是, 预测哪些列经常一起访问是非常困难的, 因此需要构建多种列组合模式的列簇, 某些列会多次出现在不同的列簇中, 从而造成存储空间浪费。考虑到星型模型的结构, 事实表很大而且列数较多, 而预测哪些列经常一起访问存在一定困难, 所以事实表采取每列单独存储方式。而用户对维表的访问, 一般是和某个维层次 (dimension hierarchy) 相关的, 因此维表采用列簇存储, 根据维表具有的维层次划分为多个列

簇, 每个列簇包含主键和对应的维层次属性。比如 SSB^[16] 中的 customer 维表可以划分为如下几个列簇: $\{[CUSTKEY, NAME], [CUSTKEY, CITY], [CUSTKEY, NATION], [CUSTKEY, REGION], [CUSTKEY, \text{其他列}]\}$ 。划分后的 customer 表如图 1 所示。

通过这种方式有效地减少了联接操作中的磁盘 I/O; 同时无论是事实表还是维表只访问需要操作的列, 提高了 CPU 缓存利用率, 使算法具有缓存敏感特性。

事实表的外键列占用存储空间远远大于维表, 比如 SSB 数据集在 $SF=10$ 时, 事实表外键列 $lo_custkey$ 占用存储空间约为 1 GB (含位置索引), 如果 Hadoop 的块大小为 64 MB, 采用 3 份复制策略, 则 $lo_custkey$ 需要 48 块存储, 可以分布到小规模集群中的大部分节点。而维表 customer 的大小约为 28 MB, Hadoop 会采用一个数据块存储, 最多只能分布在 3 个节点上。因此对事实表和维表采取不同的复制策略, 要求维表的复制数至少为 $n_d = \text{size}(fk_i) / 64$ 。另外, 星型模型中事实表的每个外键列一般情况下只需要与对应的维表进行联接操作, 事实表和维表采取相关性存储, 也就是事实表外键列和对应维表的数据块存储在同一节点, 从而减少联接操作过程中从其他节点拉取数据的情况, 减少网络 I/O。存储格式可以用图 2 表述, 其中 D_{2-m} 表示维表列簇块, fk_i 表示外键列的分块。

3.2 算法描述

基于上述存储结构, 本文提出了 CC-MRSJ 算法。将星型联接划分为两个步骤完成: 首先每个维表和对的事实

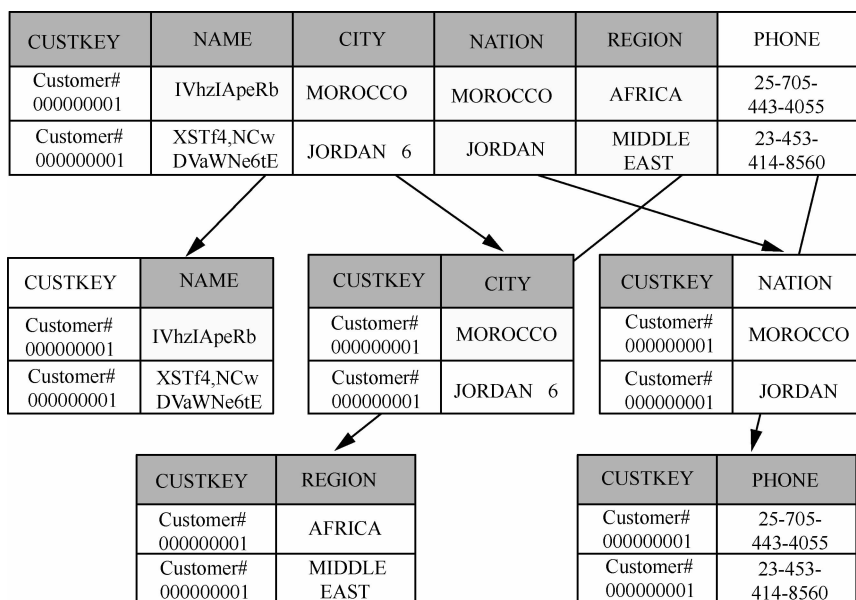


图 1 维表的列簇存储模式

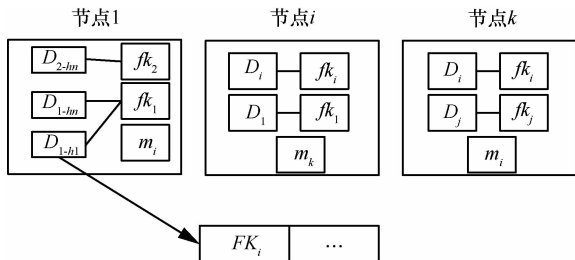


图2 星型模型存储模式

表外键列进行联接;然后对产生的中间结果进行联接并读取测度列的值,从而得到最终结果。执行过程如图3所示。

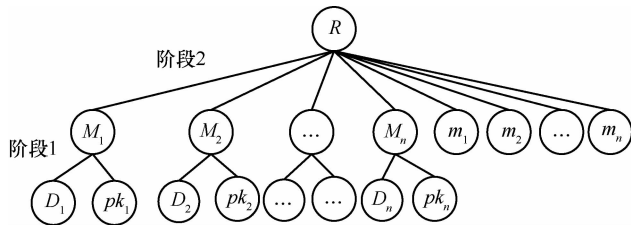


图3 星型联接执行计划

可以用式(1)表示:

$$R=(fk_1 \bowtie D_1) \bowtie (fk_2 \bowtie D_2) \dots \bowtie (fk_n \bowtie D_n) \bowtie (m_1 \dots m_n) \quad (1)$$

在算法的第一个阶段中,维表和外键列的联接操作根据集群中的计算资源可以并行或串行。所谓并行,就是多个外键列和对应的维表同时进行联接;串行是一次一个维表和一个外键列联接。通过并行可以进一步提高算法的效率。

根据维表的存储和分布特点,联接方式可以是

map-reduce 联接,也可以是 map 联接。如果维表的全部数据可以在一个节点获得,则外键列的一部分与维表联接,使联接可以在 map 阶段完成,否则需要 map-reduce 联接才能完成。

这一阶段的输出结果是以位置索引为 key 的键值对,值为联接需要从维表中读取的某列的值。这一阶段是 CC-MRSJ 算法的主要代价。

第二个阶段对位置索引进行排序,然后计算每个位置索引的个数(count),如果个数与参加星型联接的维表个数相等,则将此位置索引对应的值组合输出。对于测度列,根据位置索引以随机访问的方式读取对应位置的值。

下面以 SSB 数据集中的 Q3.1 为例,说明算法的执行过程,Q3.1 涉及多个查询条件,是一个事实表和 3 个维表的联接。只考虑联接操作,因此去掉了 group by 操作和聚集计算,该查询语句具有如下形式:

```
SELECT c_nation, s_nation, d_year, lo_revenue
FROM customer
JOIN lineorder ON lo_custkey = c_custkey
JOIN supplier ON lo_suppkey = s_suppkey
JOIN ddate ON lo_orderdate = d_datekey
WHERE c_region = 'ASIA'
AND s_region = 'ASIA'
AND d_year >= 1992 and d_year <= 1997;
```

详细的执行过程如图4所示。

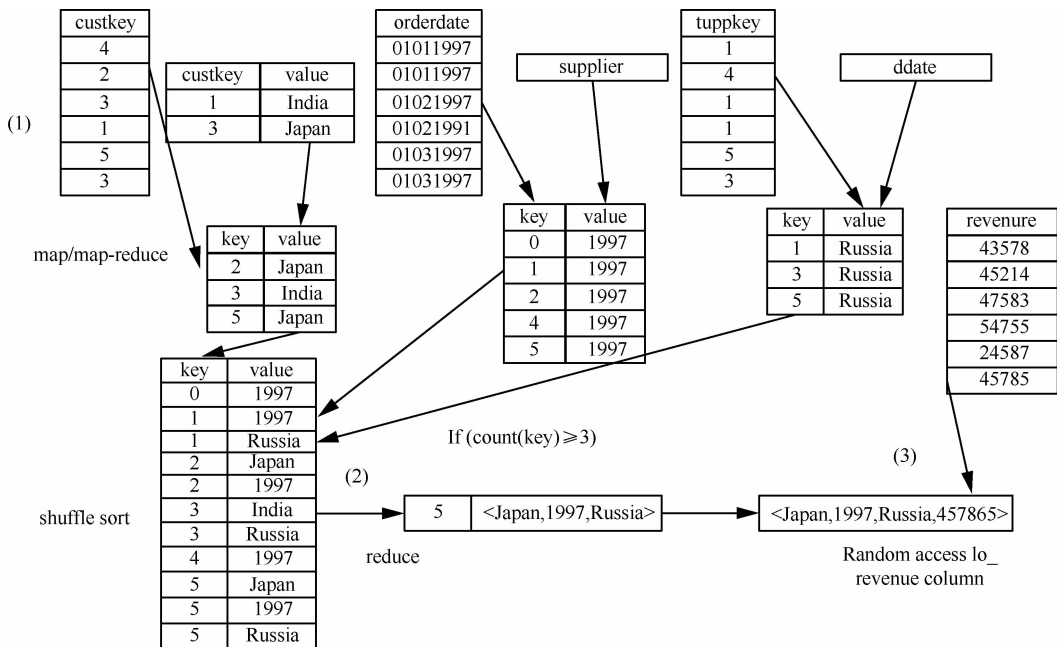


图4 星型联接的例子

首先,外键列和对应维表的列簇进行联接,生成结果的键值对是位置索引和维表中维层次的值;如图 4 所示,3 个维表分别与对应的外键列联接。

然后,每个中间结果作为输入进行 reduce 计算,根据位置索引进行排序,如果同一位置索引的值达到 3 则输出;如图 4 中位置索引为 5 的纪录满足 3 个查询条件,是最终结果集中的一条记录。

最后,根据上一步结果中的位置索引,到测度列中以随机访问的方式读取对应位置的测度值,并与结果集合并,形成联接操作的最终结果。

CC-MRSJ 算法的形式化描述如下,这里省略了算法的解释。

map (k, v)//map side join: 维表存储在分布式缓存中 DistributedCache

输入:

k is a key //position:位置索引,隐式或显式 RowId

v is a column value //foreign key

begin

for each foreign key v_i column

for each item d_j in dimension table

if ($v_i == d_j.value$)

output($k_i, d_j.value$)

end

reduce (k, v)

输入:

k is a key (join key) // position:位置索引

v is a dimension value of corresponding position

begin

1 int count=0;

for v_i in values

count++;

record = record + v_i ;

if (count==number of dimension tables in star join)

output($k_i, record$);//最终结果

end

3.3 算法分析

影响 CC-MRSJ 算法的主要因素是算法第一阶段的磁盘 I/O 和网络数据传输。其中磁盘 I/O 主要包括原始数据的读取和中间结果的写入、读取;而网络数据传输主要是 map 的结果向 reduce 节点的传输。事实表外键列所需存储

空间比维表要大很多,而且一般情况下维表和外键列联接后的结果是过滤后的数据,相对于原始数据需要更少的存储空间。因此,算法的主要代价在第一个阶段,而第一个阶段是由多个相似的事实表外键列和维表层次列簇之间的联接,而此操作具有相似的代价,所以算法的代价与联接中维表的个数,也就是外键列的个数成正比:

$$p \approx k \cdot \text{number}(fk_i) \quad (2)$$

其中, p 表示代价, k 表示比例因数。

另外,如果集群的规模足够大,可以支持多个维表和事实表外键的并发联接操作,算法可以保持近似的执行效率,即算法具有横向扩展特征。

目前,计算机普遍配置有很高的内存容量,因此可以利用大内存优化算法的性能。如果维表的列簇可以存储在节点的内存中,将维表分发到每个节点的本地内存中,则外键列和维表之间的联接可以通过 map 联接方式完成,从而获得更高的效率。Hadoop 的最新版本也提供了分布式缓存(Hadoop distributed cache)功能,为应用程序充分利用大内存提供了便利。

4 实验结果与分析

4.1 实验环境

本实验采用 SSB 数据集,SSB 数据集是由 TPC-H 数据集演变而来,其中包括一个事实表(LINEORDER)和 4 个维表(CUSTOMER、SUPPLIER、PART 和 DATE)。当设置 $SF=10$ 时,事实表大约为 6 GB,可以划分为约 100 个 64 MB 的数据块,每个数据块配置 2 个备份,对于包含有限个节点的小集群,数据可以均匀地覆盖整个集群。数据以文本文件的形式存储在磁盘上。对于事实表的每列只存储列值,在 map 操作中,输入的键值对为位置索引和对应位置的列值。

目前 Hive 算法支持星型联接,CC-MRSJ 算法与 Hive 进行比较,Hive 的版本采用 0.8.1,数据存储格式包括 TextFile 和 RCFile。

实验室搭建了包括 6 个节点的 MapReduce 测试环境,主节点配置的机器是 DELL PowerEdge T310,使用 Intel Xeon CPU X3430@ 2.40 GHz 处理器和 2 GB RAM,系统 Linux Ubuntu 版本。数据节点是 Lenovo 计算机,配置 Intel Pentium D 处理器和 2 GB RAM。采用的 Hadoop 的版本为 0.20.2。

SSB 的 Q4.1 查询语句在去掉分组和聚集运算后的格式如下:



```

SELECT c_nation, p_name, s_nation, d_year,
lo_revenue, lo_supplycost
FROM lineorder
JOIN customer ON lo_custkey = c_custkey
JOIN supplier ON lo_suppkey = s_suppkey
JOIN part ON lo_partkey = p_partkey
JOIN dwdate ON lo_orderdate = d_datekey
WHERE c_region = 'ASIA'
AND s_region = 'ASIA'
AND d_year >= 1992 and d_year <= 1997
AND (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2');

```

此查询是一个事实表和 4 个维表的联接操作。

另外, Hive 需要将 SQL 语句转化为 MapReduce 程序, 目前 Hive 转化的后程序还存在较大的优化空间, 于是一些研究人员设计了 SQL 转 MapReduce 的第三方程序, 比如 YSmart^[19,20]等, 以期提高 MapReduce 程序的效率, 但这些转化只是语句方面的优化, 并未考虑数据存储格式, 所以算法效率和 Hive 转化后的程序并没有较大提高, 所以本文直接与 Hive 系统进行比较。

4.2 实验结果

首先对 SSB 数据集上去掉聚集运算的 Q3.1 和 Q4.1 查询进行比较测试, 分别与 Hive 中的以 TextFile 和 RCFile 存储格式的联接算法进行比较, 实验结果如图 5 所示。

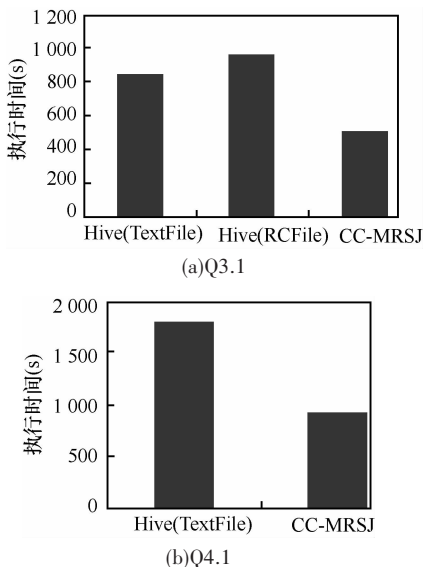


图 5 原始查询

对于星型联接, 目前 Hive 处理的方式是事实表先与其中一个维表联接, 然后中间结果和另一个维表进行联接, 直到所有维表处理完毕, 即 $((F \bowtie D_1) \bowtie D_2) \dots \bowtie D_n$, 这样会产

生很大的中间结果, 因此性能很难提高。另外, 在测试环境中 Hive 系统采用 TextFile 和 RCFile 存储格式对算法性能的影响有限, 如图 5(a) 所示, 所以在 Q4.1 的测试中省略了 RCFile 的执行时间。

对于某些联接操作, 不需要读取测度列, 因此进一步测试了星型联接中去掉测度列的情况, 测试结果如图 6 所示。

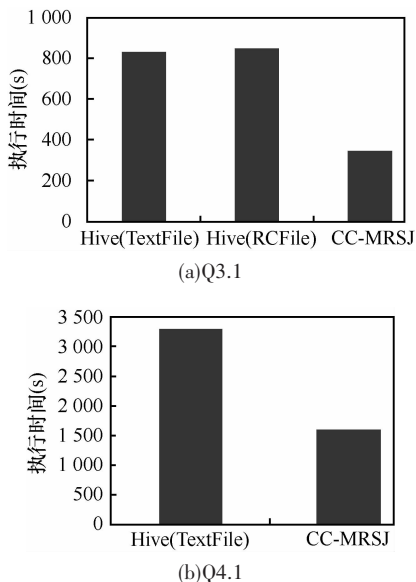


图 6 不包括测度列

由于 CC-MRSJ 算法对测度列单独处理和存储, 联接中如果没有测度列, 算法不需要访问测度列, 从而需要更少的磁盘 I/O, 而 Hive 系统无论采用 TextFile 还是 RCFile 都需要访问测度列, 从而造成额外的 I/O。因此, CC-MRSJ 获得了更高的加速比。

某些情况下, 联接查询不包括 WHERE 查询条件, 于是测试了 CC-MRSJ 将 Q3.1 和 Q4.1 中的 WHERE 条件省略后的效率, 测试结果如图 7 所示。

在这种条件下, 由于没有过滤条件, 算法需要读取的数据量比包含 WHERE 条件的联接操作要多, 所以算法执行的时间更长。但是 CC-MRSJ 通过数据划分及相关性存储和时延实例化, 从而有效地减少了 I/O, 保证了算法执行的高效率。因此, CC-MRSJ 算法同样优于 Hive 算法。

5 结束语

本文初步探讨了在 MapReduce 环境下利用合理的数据划分和组织格式, 提高星型联接算法执行效率, 算法设计过程中也充分考虑了现代处理器的缓存特性。通过实验比较, 算法与 Hive 中的联接算法相比, 获得了近似两倍的

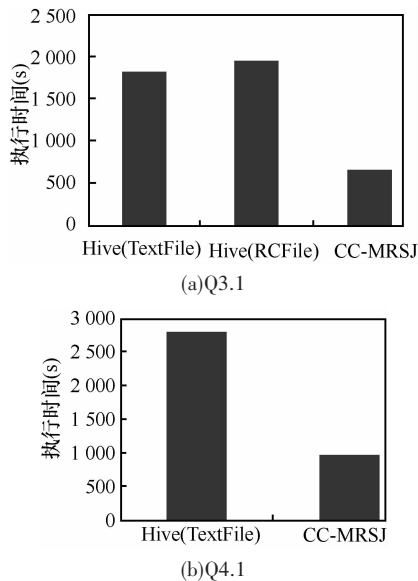


图 7 不包括 WHERE 条件

性能提升。另外,现代处理器普遍具有多核和大内存,而目前的 MapReduce 系统对此资源利用不足,因此,如何进一步利用现代处理器特征,提高 MapReduce 算法的执行效率,使 MapReduce 算法在横向扩展的同时利用处理器的纵向扩展特征将是未来的研究方向之一。

参考文献

- Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 2008(1)
- Chang F, Dean J, Ghemawat S, *et al.* Bigtable: a distributed storage system for structured data. *ACM Transactions on Computer Systems*, 2008(2)
- Thusoo A, Sarma J S, Jain N, *et al.* Hive-a warehousing solution over a MapReduce framework. *Proceedings of the VLDB Endowment*, 2009, 2(2): 1626~1629
- Gates A, Natkovich O, Chopra S, *et al.* Srivastava, building a high level dataflow system on top of MapReduce: the pig experience. *Proceedings of the VLDB Endowment*, 2009, 2(2): 1414~1425
- Stonebraker M, Abadi D J, Batkin A, *et al.* C-store: a column-oriented dbms. *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, 2005: 553~564
- Abadi D J, Madden S, Hachem N. Column-stores vs row-stores: how different are they really. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, Vancouver, 2008: 967~980
- Ailamaki A, DeWitt D J, Hill M D, *et al.* Weaving relations for cache performance. *Proceedings of the 27th International Conference on Very Large Data Bases*, Roma, 2001:169~180
- Lee R, Yin H, Zheng S, *et al.* RCFile: a fast and space-efficient data place-ment structure in MapReduce-based warehouse systems. *ICDE 2011, Hannover, Germany: 2011:1199~1208*
- Floratou A, Patel J M, Shekita E J, *et al.* Column-oriented storage techniques for MapReduce. *Proceedings of the VLDB Endowment*, 2011(7)
- Lin Y T, Agrawal D, Chen C, *et al.* Llama: leveraging columnar storage for scalable join processing in the MapReduce framework. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, Athens, Greece, 2011
- Blanas S, Patel J M, Ercegovac V, *et al.* A comparison of join algorithms for log processing in mapreduce. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* Indiana, USA, 2010:975~986
- Han H, Jung H S, Eom H S, *et al.* Yeom: scatter-gather-merge: an efficient star-join query processing algorithm for data-parallel frameworks. *Cluster Computing*, 2011, 14(2): 183~197
- Rao J, Ross K A. Cache conscious indexing for decision-support in main memory. *VLDB*, Edinburgh, Scotland, 1999:78~89
- Brewer E A. Towards robust distributed systems. *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, Portland, Oregon, 2000
- Zhang S B, Han J Z, Liu Z Y. Accelerating MapReduce with distributed memory cache. *ICPADS 2009, Shenzhen, China*, 2009: 472~478
- Shinnar A, Cunningham D, Saraswat V, *et al.* M3R: increased performance for in-memory Hadoop jobs. *Proceedings of the VLDB Endowment*, 2012(5)
- O'Neil P, O'Neil E, Chen X. The star schema benchmark, <http://www.cs.umb.edu/~poneil/star.SchemaB.PDF>, Minneapolis, 2007
- Apache Hadoop. <http://hadoop.apache.org/>, 2012
- Lee R, Luo T, Huai Y, *et al.* YSmart: Yet another SQL-to-MapReduce translator. *Proceedings of the 31st International Conference on Minneapolis, MN, USA*, 2011:25~36
- Huai Y, Lee R, Zhang S, *et al.* A matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems. *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Cascais, 2011

[作者简介]



周国亮,男,博士,华北电力大学在站博士后,主要研究方向为云计算和大数据处理技术。

朱永利,男,博士,华北电力大学教授、博士生导师,主要研究方向为人工智能及应用、网络化监控与电力系统自动化。

王桂兰,女,华北电力大学博士研究生,主要研究方向为风机故障预测。

(收稿日期:2013-05-31)