文章编号:1674 - 2869(2017)05 - 0508 - 06

消除规范关系连接冗余的二次排序算法研究

刘黎志[1,2],张 威[1,2]
1. 智能机器人湖北省重点实验室(武汉工程大学),湖北 武汉 430205;
2. 武汉工程大学计算机科学与工程学院,湖北 武汉 430205

摘 要:使用MapReduce框架对规范的一对多关系实体进行连接操作时,一方实体的各个属性会在连接的结果中产生大量冗余.通过对二次排序算法进行优化,重新定义Map阶段的分区过程、Shuffle阶段的排序及分组过程,使得Map阶段的输出为包含一方实体属性值和多方实体排序值的组合键及包含多方实体属性值的集合.Reduce阶段将组合键进行分解,提取一方实体的主码作为HBase表的行健,并将组合键中一方实体的各个属性值及多方实体属性值集合分别写入HBase表中对应的列,从而既实现了连接的语义,又消除了冗余.实验证明,优化后的算法可以消除一方实体属性值在连接结果中的冗余,提高了对连接结果的查询效率.

关键词:MapReduce;连接冗余;二次排序;HBase
中图分类号:TP311 **文献标识码**:A **doi**:10.3969/j.issn.1674-2869.2017.05.018

Secondary Sort-Based Algorithm for Eliminating Normative Join Redundancy

LIU Lizhi[1,2], *ZHANG Wei*[1,2]
1. Hubei Key Laboratory of Intelligent Robot (Wuhan Institute of Technology), Wuhan 430205, China;
2. School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan 430205, China

Abstract: The join results of two entities with normative one-to-many relationship by MapReduce may contain some redundancy of one side entity. A combination key with one side entity properties and multi-side sorted values and a list of multi-side entity properties can be got as the input of reduce stage, by optimizing secondary sort-based algorithm and redefining the partition function of map stage, sort and group function of shuffle stage. After splitting the combination key at reduce stage, the key of one side entity was extracted as rowkey of the HBase table to store the join results, and the other properties of the one side entity and the list containing multi-side entity properties were put in the corresponding columns of the HBase table, so the join semantics was realized and the redundancy was eliminated. The examination proves that the optimized algorithm can eliminate the redundancy of one side entity properties and promote the data query efficiency of the join results.

Keywords: MapReduce; join redundancy; secondary sort; HBase

MapReduce在对规范的一对多关系进行连接操作时,一方关系的各个属性值会在连接的结果中产生大量的冗余,为消除冗余,可利用HBase表的稀疏存储特性,将一方关系的各个属性值只存储一次,同时将其对应的多方关系进行按列多次存储.实现的过程可借鉴二次排序算法的思想,让一方和多方关系在Map端进行连接后,输出的Key既包含一方关系的属性,又包含多方关系中可排

收稿日期:2016-12-01
作者简介:刘黎志,硕士,副教授.E-mail:llz73@163.com
引文格式:刘黎志,张威.消除规范关系连接冗余的二次排序算法研究[J].武汉工程大学学报,2017,39(5):508-513.
LIU L Z, ZHANG W. Research on the secondary sort algorithm for eliminating normative join redundancy[J]. Journal of Wuhan Institute of Technology, 2017, 39(5): 508-513.

序的值，从而使得Reduce端在规约时可将Key包含的一方关系的属性值及多方关系的经过二次排序后的属性值直接写入HBase表.

MapReduce是Google在2004年提出一个用于处理大数据的分布式计算框架，其数据处理的流程分为Map、Shuffle及Reduce三个阶段. 在Map阶段，原始数据源根据其数据特征被划分成若干数据块，每个数据块由集群中的节点进行Map逻辑处理，结果以Key/Value（键/值对）的形式输出. Shuffle阶段负责对Key/Value对进行排序及分组，Map阶段的排序发生在将节点内存缓冲区的key/Value写入到本地磁盘spill文件，及将多个本地磁盘spill文件合并为一个spill文件时，排序的过程为：首先根据key所属的Partition（分区）排序，每个Partition再按Key进行排序. Map阶段完成后，每个的Partition会被拷贝到对应的Reduce节点，由于Reduce节点会接受来自多个Map节点的数据，故Shuffle在Reduce阶段的任务就是将来自不同Map节点的Partition按Key值进行归并排序后，将Key/Value根据Key值分组为[Key，List(Value1，Value2…Valuen)]作为Reduce任务的输入. Reduce阶段负责对[Key，List< Value1，Value2…Valuen]按特定逻辑进行规约处理，并将结果输出[1-2].

Hadoop MapReduce是Google MapReduce框架的开源实现[3]，通过对Hadoop MapReduce进行扩展，可以将HBase与MapReduce进行集成，从而使得HBase数据表和外界数据源可以以MapReduce的方式进行双向交互，从而提高数据的处理速度和效率. HBase是建立在Hadoop之上，具有高可靠性、高性能、列存储、可伸缩、实时读写特点的数据库系统，能够为海量的数据提供高性能的数据维护及查询服务[4-8]. 可以利用MapReduce将具有相同属性的文件进行连接操作，根据参与连接的文件大小可选择使用Reduce端连接、Map端连接、Semi（半）连接及Reduce端+Bloom Filter连接，连接的结果可以写入文本文件，也可以直接写入HBase数据表. 由于MapReduce的Shuffle过程默认按连接结果的Key进行排序，若需要对Value也进行排序，则需要重新定义Shuffle的排序和分组过程，进行二次排序，从而使得连接的结果首先按Key排序，然后再按Value排序[9-15].

1　改进的二次排序算法

假设规范的一方关系为M(MKEY，MATT1，MATT2，…，MATTn)，多方关系为S(SKEY，MKEY，SATT，SVALUE)，其中MKEY为关系S的外码，(mkey[m]，matt1[m]，matt2[m]，…，mattn[m])，$m\in[1,n]$表示关系M的一个元组，关系S中的SATT属性的取值范围为{satt1，satt2，…，satt$n$}，SValue的取值范围为{svalue1，svalue2，…，svaluen}，且按序号从小到大排序的整型值. 则使用MKEY对关系M和S进行连接操作，并根据SVALUE值进行二次排序后的结果如下所示：

(mKey[1]matt1[1]matt2[1])…mattn[1])satt1 svalue1
(mKey[1]matt1[1]matt2[1])…mattn[1])satt2 svalue2
(mKey[1]matt1[1]matt2[1])…mattn[1])satt3 svalue3
(mKey[1]matt1[1]matt2[1])…mattn[1])satt4 svalue4
(mKey[2]matt1[2]matt2[2])…mattn[2])satt4 svalue1
(mKey[2]matt1[2]matt2[2])…mattn[2])satt3 svalue3
(mKey[2]matt1[2]matt2[2])…mattn[2])satt1 svalue4
(mKey[3]matt1[3]matt2[3])…mattn[3])satt4 svalue3
(mKey[3]matt1[3]matt2[3])…mattn[3])satt1 svalue3
…

可见一方关系M的每个元组的各个属性会在连接结果中产生了大量冗余，消除冗余的方法是将连接的结果进行转换，写入HBase的数据表. 由于HBase表是按列存储的，在定义表结构时只需要定义列族(Column Family)，对属于列族的列的数量没有限制，以ColumnFamily：Qualifier的形式表示一个列名，Qualifier可以是任意的字节数组. 因此可以以S：satt[k]列，$k\in[1,n]$，S为多方关系名，satt[k]为S中的SATT属性的在连接结果中的值，来存储连接结果中的多方关系SVALUE属性的值. 对于连接结果中的一方关系，提取mkey[m]，$m\in[1,n]$为HBase表的行健，以M：MATT[m]列，$m\in[1,n]$，M为一方关系名，MATT[m]为M中的MATT属性名，来存储一方关系在连接结果中和mkey[m]对应的(matt1[m]，matt2[m]，…，mattn[m])属性值，从而使得一方关系的连接结果只存储了一次，既实现了连接的语义，又消除了冗余. HBase存储列值时默认按列名进行排序，故经过二次排序后的连接结果的svalue[t]，$t\in[1,n]$可能不会按照排序后的次序进行存储，可增加M：Seq列存储排序后的svalue[t]值及其与satt[k]，$k\in[1,n]$的对应关系. 经过二次排序后的连接结果在HBase表中的存储结构如图1所示.

实现将一对多的规范关系进行连接，二次排序后，直接写入HBase表，其过程如下.

1.1　自定义组合键

MapReduce的Shuffle阶段只能按Key对数据进行排序，因此若需要在对Key进行排序后，再对

rowkey	Timestamp	*M*	*S*
makey[1]	142682320446	*M*:Matt 1 matt1[1]	*S*:satt1 svalue 1
		M:Matt 2 matt2[1]	*S*:satt2 svalue 2
		…	*S*:satt3 svalue 3
		M:Matt*n* matt*n*[1]	*S*:satt4 svalue 4
		M:Seq{satt1:savlu1; satt2:savlu2; satt3:savlu3; satt1:savlu1}	
makey[2]	142686720478	*M*:Matt 1 matt1[2]	
		M:Matt 2 matt2[2]	*S*:satt1 svalue 4
		…	*S*:satt3 svalue 3
		M:Matt*n* matt*n*[2]	*S*:satt4 svalue 1
		M:Seq{satt4:savlu1; satt3:savlu3; satt1:savlu4}	
makey[3]	142756320133	*M*:Matt 1 matt1[3]	
		M:Matt 2 matt2[3]	*S*:satt1 svalue 7
		…	*S*:satt4 svalue 3
		M:Matt*n* matt*n*[3]	
		M:Seq{satt4:savlu3; satt1:savlu7}	
…			

图1 二次排序连接结果的存储结构

Fig. 1 Storage structure of secondary sort join result

Value进行排序，必须使得Map阶段输出的Key包含多方关系*S*中的SVALUE值，为将一方关系*M*的各个属性写入HBase表，Key还要包含一方关系*M*的各个属性值，自定义的组合键如下所示：

```
class CombinationKey implements WritableComparable<CombinationKey>
{   Text firstKey;
  IntWritable secondKey;
    //读取及存储部分代码省略...
}
```

其中firstKey存储*M*关系中的MKEY及各个MATT属性值以字符“\t”分隔的字符串，mkey为以“\t”分隔的第一个子字符串，secoondKey存储S关系中的SVALUE值.

1.2 实现Map端连接

由于一方关系*M*的数据一般较小，故可将其数据文件复制多份，让每个map节点内存中保存一份（如存放到HashMap中），然后扫描多方关系*S*：对于*S*中的每一条记录，在HashMap中查找是否有相同的MKEY的记录，如果有，则连接后输出.在MapReduce的任务启动时，通过job.addCacheFile(“hdfs://namenode:9000/M.txt”)指定要复制的一方数据文件M.txt，JobTracker在作业启动之前会获取这个URI列表，并将相应的文件拷贝到各个TaskTracker的本地磁盘上，Map函数在重载的setup方法中通过context.getCacheFiles()可以获取到缓存到本地的文件.实现Map端连接的过程如下：

```
//定义Map的输出为< CombinationKey, Text>
HBaseMapper extends Mapper<LongWritable, Text, CombinationKey, Text>
{private HashMap<String,String>cache_M = new HashMap<String, String>();
protected void setup(Context context)
{ BufferedReader br = null; URI[] distributePaths = context.getCacheFiles();
String mInfo = null;File mFile = new File("./M.txt");

br = new BufferedReader(new FileReader(mFile.getPath())); //读缓存文件，并放到内存中
while(null!=(mInfo =br.readLine())){
String[] mParts = mInfo.split("\t"); cache_M.put(mParts[0], mInfo)} //mPart[0]为MKEY值
}
protected void map(LongWritable key, Text value, Context context)
{ string mkey = 得到S关系每一行数据的MKEY值;
string sattr =得到S关系每一行数据的SATT值;
IntWritable svalue =得到S关系每一行数据的SVALUE值;
Text mInfo = new Text(cache_M.get(mkey));
if(mInfo != null){
CombinationKey cbkey = new CombinationKey();
cbkey.setFirstKey(mInfo);cbkey.setSecondKey(svalue);
    context.write(cbkey, new Text(sattr + “\t” + svalue));
}}}
```

1.3 重新定义分区函数和排序依据及分组函数

由于在Map端就进行一方关系*M*和多方关系*S*的连接操作，故需要重新定义分区函数，使得firstKey中具有相同mkey的连接结果分到同一个区（Partition），自定义分区类的定义如下：

```
class CusPartition extends Partitioner<CombinationKey, Text>{
public int getPartition(CombinationKey key, Text value, int numPartitions) {
string mkey = 取出 key 中 firstKey 部分的 mkey; //mkey 为分组依据
return (mkey.hashCode()&Integer.MAX_VALUE)% numPartitions; }
}// numPartitions 的值为集群中 reduce 节点的数量.
```

在 Map 和 Reduce 阶段都需要对处在同一个分区的连接结果首先按 firstKey 中的 mkey 进行排序，再按 secondKey 进行排序，自定义的排序比较类定义如下：

```
class CusComparator extends WritableComparator {
public int compare(WritableComparable cbKeyOne, WritableComparable cbKeyTwo) {
CombinationKey c1 = (CombinationKey) cbKeyOne;
CombinationKey c2 = (CombinationKey) cbKeyTwo;
string mkey1 =从 c1 中取出 firstKey 部分的 mkey;IntWritabe svalue1=取出 c1 中的 secondKey;
string mkey2=从 c2 中取出 firstKey 部分的 mkey;IntWritabe svalue2=取出 c2 中的 secondKey;
if(!mkey1.equals(mkey2)) {return mkey1.compareTo(mkey2); } //以字符方式比较 mkey
else{ return svalue1-svalue2; //以数值方式比较 secondKey} }}
```

比较方法返回值值分别以小于零的值、零值、大于零的值表示小于、等于和大于.

在 Reduce 阶段将具有相同 combinationkey 的连接结果分在同一组，形成[combinationkey, List (sattr[k] "\t" svalue1, sattr[k] "\t" svalue2. . . sattr[k] "\t" svaluen)]，$k \in [1,n]$. 分组的依据仍然为 firstkey 的 mkey 部分，自定义的分组类如下所示：

```
class CusGrouping extends WritableComparator{
public int compare(WritableComparable cbKeyOne, WritableComparable cbKeyTwo) {
CombinationKey c1 = (CombinationKey) cbKeyOne;
CombinationKey c2 = (CombinationKey) cbKeyTwo;
string mkey1 =从 c1 中取出 firstKey 部分的 mkey;
string mkey2=从 c2 中取出 firstKey 部分的 mkey;
return mkey1.compareTo(mkey2); }}
```

1.4　在 Reduce 阶段将连接结果写入 HBase 表

Reduce 首先对输入的 combinationkey 进行分解，取出 firstKey 中的 mkey 作为 HBase 表的行健，然后将 firstKey 中的其它属性值依次以 M:MATT[m]，$m \in [1,n]$列存储. 对已经按 svalue 排序好的集合 List(sattr[k] "\t" svalue1, sattr[k] "\t" svalue2. . . sattr[k] "\t" svaluen)，以 S:sattr[k]，$k \in [1,n]$列存储对应的 svalue 值. 由于 HBase 默认按列的名称 S:satt[k]进行排序，故存储的次序可能与排序的结果不一致，可以增加 M:seq 列，存储排序后的 svalue 值，Reduce 的过程定义如下：

```
class SCHBTReducer extends TableReducer<CombinationKey,Text,ImmutableBytesWritable>{
public void reduce(CombinationKey key,Iterable<Text> values,Context context){
string mSeq = "";
string[] mParts = key.getFirstKey().toString().split("\t");
Put put = new Put(mParts[0].getBytes());//行健为 mkey
for(int i=1;i<mParts.length;i++){//存储一方关系 M 的各个属性值
put.add("M". getBytes(), MATT[i].getBytes(),mParts[i].getBytes())}
while (values.iterator().hasNext()) {//存储多方关系的 satt 及 svalue 值
string[]sParts = ite.next().toString().split("\t");
put.add("S." getBytes(),sParts[0].getBytes(),,sParts[1].getBytes());
mSeq+= sParts [0] +":" + sParts [1] + ";";}
put.add("M".getBytes()," mSeq".getBytes(),mSeq.getBytes());//存储按 svalue 排序的结果
context.write(new ImmutableBytesWritable(mParts[0].getBytes()), put);//写入 HBase 表
}}
```

2　实验部分

实验环境为包含有 4 个节点的 Hadoop 集群，1 个主节点，4 个数据节点（主节点也为数据节点），节点计算机的配置为 Pentium(R) Dual-Core E500 2.5 GHz，2 GiB 内存，160 GiB 硬盘. 集群使用的操作系统为 ubuntu-14.04.1-server-i386，Hadoop 版本为 2.4.1，HBase 版本为 0.98,7，客户端使用 Eclipse 4.2.1，Java SE 1.7 为开发环境. 以学生选课关系模拟规范的一对多关系，模拟数据如图 2 所示 .

在集群的 HDFS 上建立 SecSort_MFileIn 目录，上传 std.txt 文件到该目录，建立 SecSort_SFileIn 目录，上传 course.txt 文件到该目录. 在 Main 函数中对 Job 进行如下的配置：

```
String[] ioArgs = new String[] {"hdfs://MainDataNode:9000/SecSort_MFileIn/std.txt",
"hdfs://MainDataNode:9000/SecSort_SFileIn"};
job.addCacheFile(new URI(ioArgs[0]));//设置 Map 端连接需要复制到每个节点内存的一方文件
job.setJarByClass(SecSortHBase.class);//设置任务的类名
job.setMapperClass(HBaseMapper.class);//设置 Map 类名
job.setReducerClass(SCHBTReducer.class);//设 置 Re-
```

duce类名

job.setPartitionerClass(CusPartition.class); //设置自定义分区类

job.setSortComparatorClass(CusComparator.class);//设置自定义排序比较类

job.setGroupingComparatorClass(CusGrouping.class);//设置自定义分区类

job.setMapOutputKeyClass(CombinationKey.class);//设置Map输出的Key类型

job.setMapOutputValueClass(Text.class);//设置Map输出的Value类型

FileInputFormat.setInputPaths(job, new Path(ioArgs[1])); //设置任务的输入路径

job.setOutputFormatClass(TableOutputFormat.class);//设置任务输出为HBase表

job.getConfiguration().set(TableOutputFormat.OUTPUT_TABLE, "t_sc");//设置输出的表名

System.exit(job.waitForCompletion(true) ? 0 : 1); //运行任务

任务在集群中运行后,将结果写入到HBase的t_sc表中的结果如图3所示.

TableName:std 文件: std.txt

sno	sname	sgender	sbirth
201012001	Jack	M	1990-11-01
201012002	Tom	M	1991-12-11
201012003	Jenny	F	1989-04-13
201012004	David	M	1990-02-01
201012005	Anne	F	1989-04-01

TableName:course 文件: course.txt

scno	sno	course	score
0001	201012001	Maths	98
0002	201012005	Maths	77
0003	201012002	Maths	78
0004	201012001	Database	88
0005	201012002	English	88
0006	201012004	C++	88
0007	201012002	Physics	77
0008	201012003	Arts	100
0009	201012003	DataStruct	90
0010	201012001	Arts	67
0011	201012005	English	56
0012	201012001	NetWorks	77
0013	201012004	OS	65
0014	201012002	Chemistry	89
0015	201012005	Database	65

图2 实验模拟数据

Fig. 2 Experimental simulation data

```
201012001               column=std:CourseSeq, timestamp=1482233387867, value=Maths:98;Database:88
                        ;Networks:77;Arts:67;
201012001               column=std:Gender, timestamp=1482233387867, value=M
201012001               column=std:SName, timestamp=1482233387867, value=Jack
201012002               column=course:Chemistry, timestamp=1482233387867, value=89
201012002               column=course:English, timestamp=1482233387867, value=88
201012002               column=course:Maths, timestamp=1482233387867, value=78
201012002               column=course:Physics, timestamp=1482233387867, value=77
201012002               column=std:Birth, timestamp=1482233387867, value=1991-12-11
201012002               column=std:CourseSeq, timestamp=1482233387867, value=Chemistry:89;English
                        :88;Maths:78;Physics:77;
201012002               column=std:Gender, timestamp=1482233387867, value=M
201012002               column=std:SName, timestamp=1482233387867, value=Tom
201012003               column=course:Arts, timestamp=1482233387867, value=100
201012003               column=course:DataStruct, timestamp=1482233387867, value=90
201012003               column=std:Birth, timestamp=1482233387867, value=1989-04-13
201012003               column=std:CourseSeq, timestamp=1482233387867, value=Arts:100;DataStruct:
                        90;
201012003               column=std:Gender, timestamp=1482233387867, value=F
201012003               column=std:SName, timestamp=1482233387867, value=Jenny
201012004               column=course:C++, timestamp=1482233387867, value=88
201012004               column=course:OS, timestamp=1482233387867, value=65
201012004               column=std:Birth, timestamp=1482233387867, value=1990-02-01
201012004               column=std:CourseSeq, timestamp=1482233387867, value=C++:88;OS:65;
201012004               column=std:Gender, timestamp=1482233387867, value=M
201012004               column=std:SName, timestamp=1482233387867, value=David
201012005               column=course:Database, timestamp=1482233387867, value=65
201012005               column=course:English, timestamp=1482233387867, value=56
201012005               column=course:Maths, timestamp=1482233387867, value=77
201012005               column=std:Birth, timestamp=1482233387867, value=1989-04-01
201012005               column=std:CourseSeq, timestamp=1482233387867, value=Maths:77;Database:65
                        ;English:56;
201012005               column=std:Gender, timestamp=1482233387867, value=F
201012005               column=std:SName, timestamp=1482233387867, value=Anne
5 row(s) in 0.1140 seconds
```

图3 实验结果

Fig. 3 Experimental results

从实验结果看出，std.txt文件中每个学生的属性只存储了一次，每个学生所选课程存储了多次，实现了连接的语义，消除了冗余，二次排序的学生成绩可直接通过查询std:CourseSeq列得到. 因以HBase表存储连接结果，可以利用HBase提供的各类方法进行数据检索，提高了数据的查询效率.

3　结　语

利用MapReduce对具有相同属性的关系进行连接时，不可避免会产生冗余，将Map端输出的Key进行组合，使其包含产生冗余的属性，并重新定义分区、排序及分组过程得到期望的Reduce端的输入，在Reduce端将连接结果写入HBase表，可以优化规范一对多关系的连接结果冗余. 如何对多个关联的实体进行连接，实现连接的语义，又避免连接结果中不必要的冗余，将是以后研究的主要方向.

参考文献：

[1] 王珊，王会举. 架构大数据：挑战、现状与展望[J]. 计算机学报，2011，34(10)：1741-1751.
WANG S，WANG H J. Architecting big data：challenges，studies and forecasts [J]. Chinese Journal of Computers，2011，34(10)：1741-1751.

[2] 孟小峰，慈祥. 大数据管理：概念、技术与挑战[J]. 计算机研究与发展，2013，50(1)：146-169.
MENG X F，CI X. Big data management：concepts，techniques and challenges [J]. Journal of Computer Research and Development，2013，50(1)：146-159.

[3] 陈吉荣，乐嘉锦. 基于Hadoop生态系统的大数据解决方案综述[J]. 计算机工程与科学，2013，35(10)：25-35.
CHEN J R，LE J J. Reviewing the big data solution based on Hadoop ecosystem[J]. Computer Engineering & Science，2013，35(10)：25-35.

[4] LARS G. HBase：the definitive guide[M]. Sebastopol：O'REILLY，2011.

[5] ZIKOPOULOS P C，EATON C，DEROOS D，et al. Understanding big data：analytics for enterprise class Hadoop and streaming data [M]. New Youk：McGraw-Hill，2012.

[6] AIYER A，BAUTIN M，CHEN G J，et al. Storage Infrastructure Behind Facebook Messages Using HBase at Scale [J]. Bulletin of the Technical Committee on Data Engineering，2012，35(2)：996-999.

[7] VENNER J. Pro Hadoop[M]. Berkeley：Appress，2009.

[8] 蔡睿诚. 基于HDFS的小文件处理与相关MapReduce计算模型性能的优化与改进[D]. 长春：吉林大学，2012.

[9] LU W，SHEN Y Y，CHEN S，et al. Efficient processing of k nearest neighbor joins using MapReduce [J]. PVLDB，2016，5(10)：1184-195.

[10] PANSARE N，BORKAR V R，JERMAINE C，et al. Online aggregation for large MapReduce jobs [J]. PBLDB，2014，4(11)：1135-1145.

[11] OKCAN I，RIEDEWALD M. Processing theta-joins using MapReduce [C] //ACM SIGMOD International Conference on Management of Data. ACM，2011：949-960.

[12] AFRARTI F N，DAS S A，MENESTRINA D，et al. Fuzzy joins using MapReduce[C] //IEEE International Conference on Data Engineering. IEEE，2012：498-509.

[13] ZHANG X F，SHEN L，WANG M. Efficient multi-way theta-join processing using MapReduce[J]. PVLDB，2016，5(11)：1184-1195.

[14] BABU S. Towards automatic optimization of MapReduce programs[C] //ACM Symposium on Cloud Computing. ACM，2010：137-142.

[15] SILBA Y N，REED J M. Exploiting MapReduce based similarity joins [C] //ACM SIGMOD International Conference on Management of Data. ACM，2012：693-696.

本文编辑：陈小平