一种基于MapReduce的大数据集相似自连接算法

孙德才[1] 王晓霞[2]
(渤海大学信息科学与技术学院 锦州 121013)[1] (渤海大学大学基础教研部 锦州 121013)[2]

摘 要 如何快速发现数据集中重复或相似的记录是大数据处理技术中的一个基本问题。相似连接是一种有效的相似数据查找方法,且基于MapReduce的相似连接算法因对大数据集的处理能力强而得到广泛关注。通过分析当前相似连接算法进行自连接时存在的自连接冗余、读取原字符串复杂等问题,在Massjoin算法的基础上提出了一种改进的基于MapReduce的自连接算法。改进算法在过滤阶段增加了消除自身冗余的过滤条件,在验证阶段又采用了生成正反候选对和组合id等去冗余技术,并且读取原始字符串内容时只需读取数据集一次。实验数据显示,改进算法无论在过滤阶段还是在验证阶段都减少了算法的CPU时耗,结果表明所提改进策略是有效的。

关键词 相似连接,大数据,MapReduce,数据清洗
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.05.004

MapReduce Based Similarity Self-join Algorithm for Big Dataset

SUN De-cai[1] WANG Xiao-xia[2]
(College of Information Science and Technology, Bohai University, Jinzhou 121013, China)[1]
(Research and Teaching Institute of College Basics, Bohai University, Jinzhou 121013, China)[2]

Abstract How to find out duplicates/similarities in dataset is a key issue in big data processing. Similarity join is a valid operation for finding similarities, and similarity join algorithm based on MapReduce has attracted serious concern for the advantage of processing big dataset. In this paper, similarity self-join algorithms were researched and some factors which slow self-join were discovered. To accelerate self-join, an improved similarity self-join algorithm based on Massjoin was proposed. In filtration stage, new filtration criterion is added to eliminating self-join redundant pairs. In verification stage, the techniques of backward-forward pairs and combined id are adopted to eliminate more self-join redundant candidate pairs, and the dataset is scanned only once in reading original strings. The experimental results demonstrate that both filtration CPU time and the verification CPU time of new algorithm decrease. As a result, the efficiency of similarity self-join algorithm is increased by using our revision strategies.

Keywords Similarity join, Big data, MapReduce, Data cleaning

1 引言

随着互联网信息技术的迅猛发展,数据呈爆炸式增长,全球已步入了大数据时代。全球知名咨询公司麦肯锡最早提出"大数据"的概念,认为大数据是指其大小超出了典型数据库软件采集、储存、管理和分析等能力的数据集合。人们常用"4V"来描述大数据的特征[1-2],即体量巨大(Volume)、形态繁多(Variety)、生成速度快(Velocity)、价值大但密度低(Value)。

大数据时代下,如何存储、管理和分析这些大数据资源,并从中快速发现和获取有用信息是目前研究的热点问题。数据清洗是对数据的重新审查和校验,能删除重复信息,发现并纠正错误,同时保持数据的一致性,是大数据处理中的常用操作。相似连接(Similarity Join)[3-9]是一种查找数据集中相似记录对的有效方法,它在数据清理、数据集成、近似网页去重和剽窃检测等领域都具有广泛的应用。

相似连接能从一个或多个给定的数据源中找出所有满足要求的相似记录对[6],当数据源有单源时称为自连接,当数据源有两个及以上时称为多源连接。自连接找出的相似记录对都来源于同一个数据集,而多源连接的相似记录对分别来源于不同的数据集。实际应用中它根据连接结果要求的不同可分为阈值连接和Top-k连接。阈值连接指找出数据集中相似度不小于用户给定阈值的所有记录对,而Top-k连接则是指找出数据集中相似度最高的前k(用户给定)个记录对。根据数据集中记录数据的类型,相似连接又可分为集合连接、字符

到稿日期:2016-08-22 返修日期:2016-11-12 本文受教育部人文社会科学研究青年基金项目(15YJC870021,15YJC870028),辽宁省博士科研启动基金计划项目(20141138),辽宁省教育厅科学研究项目(L2015010,L2014451),辽宁省自然科学基金(2015020009),国家自然科学基金青年基金项目(61602056)资助。

孙德才(1979—),男,博士,副教授,主要研究方向为相似连接、近似串匹配等,E-mail:sdecai@163.com(通信作者);王晓霞(1977—),女,硕士,讲师,主要研究方向为相似连接、近似串匹配等,E-mail:wxxsdc@163.com。

串连接、向量连接和图等连接。根据应用中记录数据类型的不同，记录间相似度的计算函数也不同，如集合常采用 Jaccard，Dice 和 Cosine 等相似函数；字符串常采用编辑距离、汉明距离等描述。编辑距离(Edit Distance)[10]是指把一个字符串经过插入、修改或删除 3 种编辑操作转变成另一个字符串所要进行的最小操作次数。相似连接算法根据并行性可分为内存算法和并行算法两种[11]。

内存算法运行于单机上，算法允许在连接过程中访问存储在内存中的全局信息，如索引结构等。这里将其分两类进行介绍：第一类算法利用特殊的树形索引直接计算连接的结果，典型的有 TrieJoin[12]等，该类算法不适合长字符串；第二类算法则采用过滤、验证两阶段模式，如 AllPair[13]，PPJoin[14]，EDJoin[15]，VChunkJoin[16]，PartEnum[17] 和 PassJoin[18]等，该类算法先使用过滤条件(长度过滤、位置过滤、多匹配过滤或前缀过滤等)去除大量的一定不满足连接要求的记录对，然后再对通过过滤的候选记录对进行精确验证来计算其是否为真实结果。

并行算法运行在集群上，算法因并行执行而使得各节点间共享信息困难。MapReduce[6,8,11]是 Google 提出的一个高效的分布式编程框架，在大数据处理中被广泛使用。MapReduce 程序运行于多节点的集群中，该框架主要包括 map，shuffle 和 reduce 3 个阶段。各阶段间的输入、输出都是〈key，value〉对。程序员只需关注 map 和 reduce 两个关键函数的设计即可，无需关注分布式存储和并行计算调度等相关问题。目前，基于 MapReduce 的相似连接算法也得到了众多学者的关注，如，Vernica 等[19]实现了一种基于 MapReduce 的集合相似连接算法，该算法使用了前缀过滤技术，在 map 阶段的处理过程中用字符串前缀中的 token 作为 key，用字符串本身作为 value；在 reduce 阶段的处理中发现共享相同前缀 token 的字符串即为候选对。MetWally 等[20]提出了一种二阶段的 V-SMART-Join 算法，与 Vernica 等的算法相同，其也采用单 token 作为 key。Afrati 等[21]提出了多种相似连接算法，并详细分析了算法的 map 负载、reduce 负载和传输消耗等问题。Lin 等[22]针对内存算法 PassJoin[18]进行改进升级，实现了一个名为 PassJoinKMR 的基于 MapReduce 的相似连接算法，并为减少集群的负载提出了改进算法 PassJoinKMRS。同时，Deng 等[11]也基于 PassJoin 算法在 MapReduce 上实现了 MassJoin 算法，该算法通过合并相同 key-value 对，既减少了数据的传输量，又未降低算法的过滤能力，该算法不仅支持字符串连接，还被扩展到了集合连接。

本文分析了目前基于 MapReduce 的相似连接算法的相关研究，并基于 MassJoin 算法提出了相应的改进的自连接算法。本文第 2 节介绍了几种经典的相似连接算法；第 3 节详细描述了本文的改进算法；第 4 节通过实验对比了算法改进前后的性能差异；最后总结全文。

2　相关工作

2.1　Passjoin 算法

PassJoin[18]是一种基于编辑距离的字符串相似连接算法，它在 EDBT2013 的国际相似连接比赛中获得了冠军。该算法是一个内存算法，其核心是在过滤阶段生成了高质量的字符串子串，使得满足要求的串对至少共享一个子串。算法通过精确选择子串过滤了大量的无关串对，其流程如下：

(1)S_l 表示长度为 l 的字符串集合，S_l^i 表示 S_l 中第 i 个块的集合，对每个 S_l^i 创建倒排索引 L_l^i，$L_l^i(w)$表示第 i 个块是 w 的字符串集合，初始状态 L_l^i 为空。

(2)算法首先对所有字符串按长度从小到大进行排序，然后从小到大依次访问。

(3)对于一个串 s，首先把 s 分割成 $\tau+1$ 个部分，使得前面块的长度为$\lfloor |s|/\tau+1 \rfloor$，后面 $k=|s|-\lfloor |s|/\tau+1 \rfloor * (\tau+1)$块的长度为$\lceil |s|/\tau+1 \rceil$。

(4)针对每个分块分别采用子串选择算法生成一个子串集合，如第 i 个分块子串集合记为 $W(s,L_l^i)$。

(5)对于每个选择的子串 $w \in W(s,L_l^i)$，在索引中查找 $L_l^i(w)$，对于每个 $r \in L_l^i(w)$的串，调用验证算法验证 r 和 s 是否是匹配对。

(6)处理完串 s 的所有分块后，把 $\tau+1$ 个分块分别插入到对应的索引中，如第 i 个分块插入到 $L_{|s|}^i$ 中。

(7)转至步骤(3)处理下一个字符串，直到所有的字符串都处理完毕。

该算法中的核心为子串选择算法(详见文献[18])，通过长度过滤、移位过滤、位置过滤以及多匹配过滤等多种方法对可选择的子串进行过滤，剔除了大量无效子串，提高了算法的效率。

2.2　PassjoinKMR 算法

Lin 等[22]基于 MapReduce 实现并改进了 PassJoin 算法[18]，称为 PassJoinKMR 算法。该算法把字符串分割成 $\tau+k$ 个子串，并对含有 k 个以上正确匹配子串的串对进行验证，文中实验发现 $k=1$ 时算法效率最高。该算法不需要先对字符串进行排序，其算法流程如下。

(1)map 过程：针对串 s，首先把 s 拆分成 $\tau+k$ 个块，然后分别生成索引串和匹配子串。

生成索引子串：每个块都生成索引子串，即生成〈key，value〉对。用长度 $|s|$、块号 i 和索引子串内容共同组合成 key，将串 s 的 ID 和类型标识(IIFlage)作为 value。

生成匹配子串：通过子串选择(同 PassJoin)生成每个块的匹配子串，即生成〈key，value〉对。同样用长度 $|s|$、块号 i 和匹配子串内容共同组合成 key，将串 ID 和类型标识(SSFlage)作为 value。

(2)reduce 过程：拆分每个 key 对应的 value 列表，根据类型标识得到 IIlist 和 SSlist，对每个串对〈rid，sid〉($rid \in IIlist, sid \in SSlist$)进行验证，判断它们间的编辑距离是否满足匹配要求。

2.3　MassjoinBasic 算法

Deng 等[11]也在 MapReduce 框架上实现了 PassJoin 算法[18]，称为 Massjoin 算法。该算法不仅支持字符串连接，还被扩展到了集合连接。文献中的 Basic 算法严格实现了 passjoin 算法，该算法分为 3 个阶段：过滤阶段、验证阶段 1 和验证阶段 2。MassjoinBasic 算法的相似连接过程如下：

(1)过滤阶段:在 Map 过程,生成索引子串(过程参见 Passjoin 算法的过程(3))和匹配子串(采用 Passjoin 算法的子串选择算法进行子串选择得到的串)。生成的〈key,value〉对中 key 为子串、串长和块号的组合,而 value 则为字符串的编号 rid(索引子串)或 sid(匹配子串)。在 Reduce 过程,把相同 key 的 value 集合拆分成 *list*(*rid*)和 *list*(*sid*),并输出每个匹配子串 *sid* 匹配到的 *list*(*rid*)以供下阶段使用。

(2)验证阶段 1:该阶段的主要功能是读取 S 集。在 Map 过程,若读取的是过滤阶段结果,则原样输出〈*sid*,*list*(*rid*)〉;若读取的是 S 集,则输出串编号及对应串内容〈*sid*,*s*〉。在 Reduce 过程首先区分相同 *sid* 下的串内容 *s* 和匹配到的串列表 *list*(*rid*),并用串内容 *s* 替换串编号 *sid*,输出〈*s*,*list*(*rid*)〉。

(3)验证阶段 2:该阶段的主要功能是读取 R 集和验证候选对。在 Map 过程,若读取的是验证阶段 1 的结果,则拆分 *list*(*rid*)并反向输出〈*rid*,*s*〉;若读取的是 R 集则输出串编号及对应串内容〈*rid*,*r*〉。在 Reduce 过程首先区分相同 *rid* 下的串内容 *r* 和匹配到的串列表 *list*(*s*),然后对每个〈*r*,*s*〉对采用验证算法进行验证,并输出相似度满足要求的串对及相似度〈〈*r*,*s*〉,*sim*(*r*,*s*)〉。

3 基于 Massjoin 的改进自连接算法

3.1 现有算法进行自连接的不足

PassjoinKMR 是一个自连接算法,在 map 阶段产生子串,在 reduce 阶段发现候选对,然后从数据集中读取相应原始串(参见文献[22]中算法 2)以供验证。字符串集是海量数据,从中读取一个指定串需要遍历数据集,时间消耗较大,原始字符串的读取是 PassjoinKMR 算法的瓶颈。

Massjoin 算法既支持双源连接也支持自连接,例如文献[11]描述了当设置输入的两个集合 R 和 S 完全相等时,Massjoin 算法进行的是自连接。该算法的验证阶段巧妙地通过两步读取(参见文献[11]中的算法 1)解决了验证中原始字符串读取的瓶颈问题。但该算法进行自连接时存在多次读取数据集、自身连接冗余等问题。例如,自连接时在过滤阶段生成索引串时读取 R 集 1 次,而生成匹配串时读取 S 集 1 次;在验证阶段 1 中又读取 S 集 1 次,在验证阶段 2 中又读取 R 集 1 次;因该算法进行的是自连接且此时 R 集与 S 集完全相等,故自连接的输入集共被读取了 4 次,浪费了大量时间。并且自连接时算法也因无法区分 R 集和 S 集中的同一字符串而避免了本身和本身的相似计算,因此也存在自身连接冗余运算。

3.2 改进的自连接算法 Self-MassJoinBasic

本文以字符串相似连接为例,首先充分利用 Massjoin 算法的优势,并针对 MassjoinBasic 算法进行自连接存在的问题进行了改进,实现了一种基于 MapReduce 的自连接算法,称为 Self-MassJoinBasic。新算法包含 3 个阶段:过滤阶段、验证阶段 1 和验证阶段 2,每个阶段都是一个 map/reduce 过程。Self-MassJoinBasic 算法的框架图如图 1 所示。

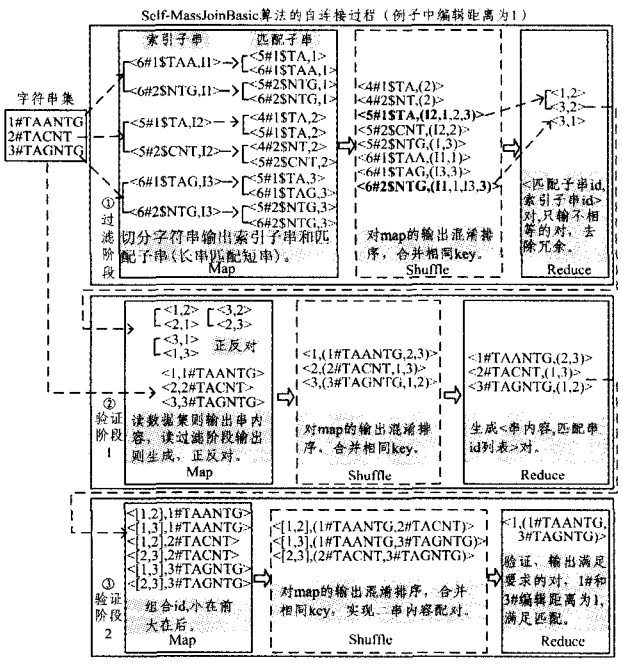

图 1 Self-MassJoinBasic 算法的框架图及例子

过滤阶段的主要目的是根据字符串的拆分和匹配信息进行过滤,并获得候选对。在该阶段算法将抛弃那些一定不满足相似要求的串对。验证阶段的主要目的是针对过滤阶段过滤结果中的候选对,先读取原始串内容,然后计算它们之间的真实相似度,最后去除那些虽然通过了过滤但确实不满足要求的候选对。改进算法的验证阶段分为两个阶段:验证阶段 1 和验证阶段 2。下面将详细介绍改进算法各阶段的处理过程。

3.2.1 过滤阶段

过滤阶段的输入为字符串集,输出为过滤后得到的候选对,其 Map,Shuffle 和 Reduce 过程如下。

1)Map 过程:过滤过程将读取字符串集,算法运行中会产生众多 map 任务,每次 map 将处理一个字符串。一次 map 中,先对读取的字符串采用 Passjoin 算法的分块方法进行分块,得到用于索引的索引子串(I 型),如图 1 中①的 map 中左边的串,伪代码见算法 1 第 2、3 行,生成的 key/value 对中用长度、块号、分块一起组合为 key,而用串 id 作为 value(图 1 例子中在 id 前面加 I 来区分两种类型的子串);然后,采用 Passjoin 的子串选择算法针对每块分别生成用于匹配的匹配子串(S 型),如图 1 中①的 map 中右边的串,伪代码见算法 1 第 4—7 行,在生成的 key/value 对中同样用长度、块号、分块一起组合为 key,并用串 id 作为 value。伪代码中第 7 行的上限由 L_u 改进为 s,能去除自连接的冗余匹配子串,例如有 a 串和 b 串,a 串会产生所有能匹配到 b 串的匹配子串,而 b 串也会产生所有能匹配到 a 串的匹配子串,本算法中通过控制长串匹配短串来减少输出的匹配子串数量,从而减少负载。另外,Map 中字符串集只被读取一次,相比 Massjoin 算法进行自连接时减少了读取字符串集的次数。

2)Shuffle 过程:在 map/reduce 框架中,在 shuffle 过程将对所有记录按 key 值进行混淆、排序,然后把 shuffle 的结果

作为 reduce 过程的输入，如图 1 例子中①的 shuffle 结果。

3）Reduce 过程：算法运行中也有众多的 reduce 任务，每次 reduce 过程处理 shuffle 结果中的一个 key/value 对。Reduce 读取的 key/value 对中 value 值为索引子串 id 值和匹配子串 id 值组成的列表。首先需把该列表拆分成索引子串 id 列表和匹配子串 id 列表，若此时这两个列表中有空表，则代表此次 reduce 处理中无匹配对，如图 1 例子中①shuffle 结果中第 1，2，5 行无匹配对；而当两个列表都不为空时才用列表元素对应组合生成 key/value 对，key 选取匹配子串列表中的 id 值，而 value 则只选择索引子串列表中值不等于此 key 的 id 值。如此能避免字符串本身和本身的配对，如图 1 例子中① shuffle 结果中第 4，6-7 行被抛弃，第 3 行生成〈3，2〉，〈1，2〉，伪代码见算法第 10－12 行，从而解决了 Massjoin 算法进行自连接时存在的自身冗余问题。

3.2.2　验证阶段 1

验证阶段 1 的输入有两组数据：字符串集和过滤阶段结果。该阶段的主要功能是提取原始串内容和去重候选对。

1）Map 过程：在 map 中，若读取的是字符串集中的串，则直接输出以串 id 作为 key、以串内容作为 value 的 key/value 对，见算法第 14－15 行。若读取的是过滤阶段结果，则产生正反候选对，如图 1 中②例子中 map 的前 4 行。生成正反候选对的目的是方便后续处理中串 id 和串内容的配对，伪代码见算法第 16－18 行，从而避免了字符集的再次读取，相比 Massjoin 算法进行自连接时再次减少了读取字符串集的次数。

2）Shuffle 过程：shuffle 过程会对所有 map 的输出 key/value 对按 key 进行混淆和排序，并将结果作为 reduce 的输入，如图 1 中②的 shuffle 过程。

3）Reduce 过程：在 reduce 过程中，每次处理一个 shuffle 输出结果的 key/value 对，此时 key 为某个串的 id，而 value 中既包含了该串的内容，又包含了该串匹配到的串的 id 列表。首先拆分列表并区分串内容和匹配到的串 id 列表，并输出新的 key/value 对，以串内容作为 key，并用去重后的 id 列表作为 value，如图 1 中②的 reduce 结果，伪代码见算法 1 的第 20－22 行。

3.2.3　验证阶段 2

验证阶段 2 的输入数据是验证阶段 1 的输出结果，该阶段的主要功能是去除冗余和详细验证候选对，从而得到最终的匹配对。

1）Map 过程：在 map 过程中，先读取验证阶段 1 输出结果中的候选对，然后生成新的 key/value 对，用原 key 中串 id 和列表中的 id 连接成组合 id 作为 key，组合时需保持值小的 id 在前，值大的 id 在后。这样做的目的是让同一候选对 shuffle 只产生一个 key/value 对，既能去掉〈id1，id2〉和〈id2，id1〉这种正反向的冗余，又能方便串内容的配对，如图 1 中③的 map，伪代码见算法 1 第 24－25 行。

2）Shuffle 过程：shuffle 过程将对 map 输出的所有 key/value 对按组合 key 进行混淆和排序，并把结果作为 reduce 的输入，如图 1 中③的 shuffle 过程。

3）Reduce 过程：在 reduce 过程中，shuffle 结果的每个 key/value 对都是一个候选对，key 为两串的组合 id，而 value 为两串内容组成的列表。首先拆分 value 得到两串的内容，然后采用 *on-line* 算法计算两串间的编辑距离，这里采用了文献[18]中的编辑距离计算方法。此时，若编辑距离满足要求则输出该匹配对，如图 1 中③的 reduce 结果，伪代码见算法 1 第 27－29 行。

算法 1　Self-MassJoinBasic

```
//s 是字符串集 S 中的一个串，id 是 s 在 S 中的编号。
//1. 过滤阶段
1. Map(〈sid,s〉) //map 处理过程
2.   for 1≤i≤Δ do //索引子串分割方法
3.     emit(〈〈s[p_i^ℓ, l_i^ℓ], i, ℓ=|s|〉, sid〉); //用串 s 生成索引子串(I 型)
4.   for L_o≤ℓ≤|s| do //改进子串选择算法
5.     for 1≤i≤Δ do
6.       for ⊥_i^ℓ≤x_i^ℓ≤⊤_i^ℓ do
7.         emit(〈〈s[x^ℓ_i, l_i^ℓ], i, ℓ〉, sid〉); //用串 s 生成匹配子串(S 型)
8. Reduce(〈sig,list(sid)〉) //reduce 处理过程，sig 为索引子串/匹配子串
9.   把 list(sid)按 I 型和 S 型分为两组，list(Isid)和 list(Ssid);
10.  foreach Ssid∈list(Ssid) do
11.    newlist(Isid)=remove(list(Isid),Ssid); //去重，删除 list(Isid)中等于 Ssid 的 Isid
12.    output(〈Ssid,newlist(Isid)〉); //生成候选对
//2. 验证阶段 1
13. Map(〈sid,list(sid)〉/〈sid,s〉)
14.  if 〈sid,s〉 then //读的是字符串集，输出 id+串内容
15.    emit(〈sid,s〉);
16.  if 〈sid,list(sid)〉 then //读的是过滤阶段输出结果，输出正反候选对
17.    for each sid1 in list(sid) do
18.      emit(〈sid,sid1〉);emit(〈sid1,sid〉);//生成正反对，用于匹配串内容
19. Reduce(〈sid,list(sid/s)〉)
20.  拆分 list(sid/s)，获取该 sid 的串内容 s 和对应串列表 list(sid);
21.  newlist(sid)=remove(list(sid));//去重，删除 list(sid)中存在的重复 sid
22.  output(〈sid&s,newlist(sid)〉);
//3. 验证阶段 2
23. Map(〈sid&s,list(sid)〉)
24.  for each id in list(sid) do
25.    emit(〈sid&id/id&sid,sid&s〉);//key 为 id 和 sid 的组合 id，组合时保证小数字在前，去重
26. Reduce(〈id1&id2,list(sid&s)〉)
27.  拆分 list(sid&s)，获得 id1 的串内容 s1 和 id2 串内容 s2;
28.  if ED(s1,s2)≤t then //验证候选对
29.    output(〈id1&s1+id2&s2,ED(s1,s2)〉); //输出符合要求的匹配对
```

对改进算法进行描述和分析，并将其与文献[11]中的 MassJoinBasic 算法进行对比，Self-MassJoinBasic 算法的主要改进点如下：

(1)通过消除自连接冗余来减少过滤阶段的数据处理量,均衡负载,在过滤过程中改进了匹配子串选择算法,力图减少产生的匹配子串数量。

(2)为减少数据集读取次数,在过滤阶段只读取字符串集一次,而在验证阶段也只读取字符串集一次,并结合生成正反候选对和组合 id 的方法实现串 id 和串内容的配对。

(3)在验证阶段采用了组合 id 的方式来进一步消除重复的正反对,进一步减少了算法的验证时间。

4 实验

4.1 实验环境

为对比算法改进前后的性能差异,本文在 hadoop 平台上实现了文献[11]中的 MassJoinBasic 算法,同时也实现了本文对应的改进算法 Self-MassJoinBasic。本文所有实验都在同一 hadoop 集群下进行的。实验集群中节点总数为 4 个,即 1 个主节点和 3 个从节点;节点硬件配置:CPU i5 4590,内存 16GB,硬盘 1TB;集群软件配置:操作系统 Ubuntu 15.10 64 位,Java 1.7,Hadoop 平台版本 1.2.1;开发环境:eclipse(indigo 版)。

本文实验中有两个测试数据集,具体情况如表 1 所列。数据集 DBLP[1] 是计算机领域以作者为核心的一个计算机类英文文献的集成数据库,实验把数据集中的记录转化成只含有作者(Author)和标题(Title)的字符串。GBEST 数据集为 NCBI GenBank 的表达序列标签(Expressed Sequence Tags, http://www.ncbi.nlm.nih.gov/),实验中去除 EST 中的描述信息,只保留了序列本身。

表 1 实验数据集

	Size/MB	Cardinality	Avg Size/Len	$\|\Sigma\|$
DBLP	95	1000000	90.7	96
GBEST	182	538160	347.1	5

4.2 算法性能对比

相似连接算法的性能指标中最重要是连接时间,map/reduce 数量、资源抢占等因素使得任务的运行时间并不能很好地衡量算法的执行效率。hadoop 任务运行的 CPU 时间(CPU time spent)是 hadoop 运行期间所有 task 任务对应进程的用户 CPU 时间和内核 CPU 时间的总和,是衡量任务计算量的准确度量。文献[11]中的 MassJoinBasic 算法和本文的改进算法在相似连接过程中都主要分为两个阶段:过滤阶段和验证阶段。过滤阶段的过滤效果的好坏也影响着验证阶段的执行效率。

相似连接算法在连接中给定的编辑距离是限定符合相似要求的两串间相似程度的参数。给定不同的编辑距离,算法的执行效率也随之改变。本实验针对 MassJoinBasic 和 Self-MassJoinBasic 算法,在 DBLP 实验数据集上分别采用编辑距离 0,2,4,6,8 进行相似连接测试,在 GBEST 实验数据集上分别采用编辑距离 0,2,4,8,12,16 进行相似连接测试。实验中统计了各算法在过滤阶段和验证阶段上的 CPU 时间消耗,各算法的性能对比如图 2 和图 3 所示。

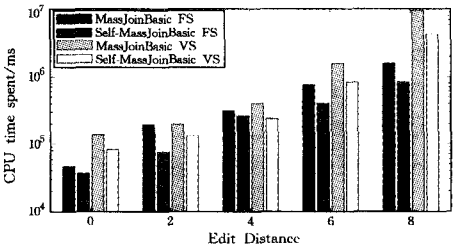

图 2 DBLP 数据集

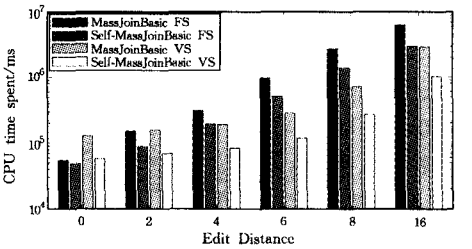

图 3 GBEST 数据集

MassJoinBasic 和 Self-MassJoinBasic 算法的过滤阶段都是一个 map/reduce 过程。它们的验证阶段都为两个连续的 map/reduce 过程,MassJoinBasic 分别进行 2 次库读取和验证操作,而 Self-MassJoinBasic 则分别进行 1 次库读取、去冗余和验证操作。由图 2、图 3 可知,对于两个数据集,改进算法无论是在过滤阶段还是在验证阶段的执行效率都有所提高,即在不同的编辑距离下过滤时间和验证时间都减少了。这是因为改进算法在过滤阶段根据自连接的特点采用了改进的匹配子串选择算法,消除了自连接冗余,减少了 map 过程生成的中间数据量和过滤阶段的输出量,并且 map 中只读取了一次字符串集,从而减少了过滤时间;而在验证阶段,改进算法又通过生成正反候选对和组合 id 等方式过滤掉了部分自连接冗余对,且在读取原始字符串内容时也仅读取数据集一次,因此验证时间也得以减少。

实验中还统计了编辑距离等于 8 时改进算法与原算法在两个实验数据集上过滤阶段的输入、中间输出和输出的数据量,统计结果如表 2 所列。

表 2 算法过滤阶段处理数据量对比($ED=8$)

		File Input Format Counters (Bytes Read)	Map output materialized bytes(Reduce Shuffle Bytes)	File Output Format Counters(Bytes Written)
DBLP	MassJoin Basic	199161974	13103938518	4078946038
	Self-Mass JoinBasic	99580987	6843056038	1210512629
GBEST	MassJoin Basic	383090562	14831022854	83920123
	Self-Mass JoinBasic	191545281	7968094237	5528029

过滤阶段的输入量(File Input Format Counters)是算法过滤时从 HDFS 读取的数据总量,改进算法因为仅进行了一次数据集读取操作,所以其输入量为原算法的一半,因此数据

[1] http://dblp.uni-trier.de/xml/

读取时间更少。在过滤阶段 map 任务的中间输出量(Map Output Materialized Bytes)是 map 阶段的输出结果，该数据经过 hadoop 的 shuffle 阶段进行混淆和排序处理后作为 reduce 阶段的输入，该数据量的大小决定了 shuffle 阶段的执行效率，从表 2 中可知，改进算法的中间输出量通过消除自连接冗余而得以减少。过滤阶段的输出量(File Output Format Counters)是算法过滤阶段的最终输出，是候选对的集合。若不考虑数据的输出格式，该数据量的大小能说明算法过滤阶段的过滤能力。从表 2 的对比可知，改进算法的候选对集更小，这也有助于算法验证时间的减少。

实验最终统计了各算法在两个实验数据集上采用不同编辑距离的总 CPU 时间，性能对比如图 4 和图 5 所示。

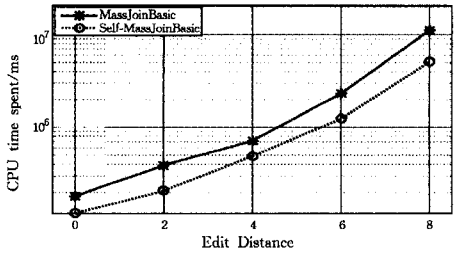

图 4　DBLP 数据集上不同编辑距离下的总 CPU 时间对比

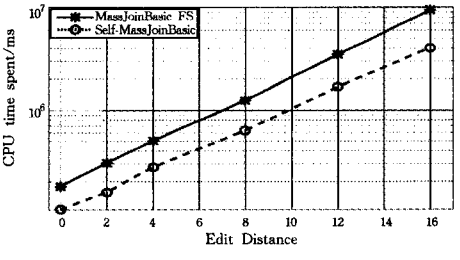

图 5　GBEST 数据集上不同编辑距离下的总 CPU 时间对比

由图 4、图 5 可知本文提出的改进思路是有效的，改进算法在不同数据集上不同编辑距离下的总 CPU 时间较原算法都有所减少。改进算法在过滤阶段通过读取数据集一次、大匹配子串 id 对小索引子串 id 的候选对生成方式等减少了过滤时间，在验证阶段通过读取数据集一次、生成正反候选对和组合 id 等也减少了验证时间，进而减少了总时间。

结束语　本文的主要研究内容是数据清理中的大数据集相似连接技术，针对当前自连接算法存在的验证字符串读取瓶颈、自身冗余、数据集重复读取等问题，提出了一种基于 MapReduce 的改进算法，即 Self-MassJoinBasic。文中详细描述了 Self-MassJoinBasic 算法进行自连接的各个改进点及实现过程。通过实验对比可知，改进算法无论在过滤时间和验证时间上都有所减少，加快了算法的连接速度。

改进算法虽然在过滤阶段和验证阶段都减少了时间，但从实验中发现算法在过滤过程中生成的子串集非常庞大，尤其是当编辑距离较大时，这样严重降低了算法的总体速度。下一步将研究减少过滤阶段生成子串数量的方法，拟通过减少过滤时间的同时允许少许增加验证时间的方式来加快算法的总体连接速度。

参 考 文 献

[1] YU W, LI S J, YANG S, et al. Automatically Discovering of Inconsistency Among Cross-Source Data Based on Web Big Data [J]. Journal of Computer Research & Development, 2015, 52 (2): 295-308. (in Chinese)
余伟，李石君，杨莎，等. Web 大数据环境下的不一致跨源数据发现[J]. 计算机研究与发展，2015，52(2)：295-308.

[2] XU J, WANG G Y, YU H. Review of Big Data Processing Based on Granular Computing [J]. Chinese Journal of Computers, 2015, 38(8): 1497-1517. (in Chinese)
徐计，王国胤，于洪. 基于粒计算的大数据处理[J]. 计算机学报，2015，38(8)：1497-1517.

[3] M Y Z, M X F, W S Y. Parallel similarity joins on massive high-dimensional data using MapReduce[J]. Concurrency and Computation: Practice and Experience, 2016, 28(1): 166-83.

[4] SILVA Y N, PEARSON S S, CHON J, et al. Similarity Joins: Their implementation and interactions with other database operators[J]. Information Systems, 2015, 52(8-9): 149-162.

[5] SANTOS L F D, OLMES CARVALHO L, OLIVEIRA W D, et al. Diversity in similarity joins[C]// Proceedings of the 8th International Conference on Similarity Search and Applications (SISAP 2015). Glasgow, UK: Springer, 2015: 42-53.

[6] PANG J, YU G, XU J, et al. Similarity Joins on Massive Data Based on MapReduce Framework[J]. Computer Science, 2015, 42(1): 1-5, 27. (in Chinese)
庞俊，于戈，许嘉，等. 基于 MapReduce 框架的海量数据相似性连接研究进展[J]. 计算机科学，2015，42(1)：1-5，27.

[7] WANG H Y, YANG L H, LIU X Q. Optimizing Top-k Similarity Join Algorithm[J]. Journal of Software, 2016, 27(12): 3051-3066. (in Chinese)
王洪亚，杨利宏，刘晓强. Top-k 相似连接算法性能优化研究[J]. 软件学报，2016，27(12)：3051-3066.

[8] CHEN Z J, ZHANG J N, LIU W Y. Region-based Spatial-textual Similarity Join in MapReduce [J]. Journal of Chinese Computer Systems, 2015, 36(10): 2245-2251. (in Chinese)
陈子军，张娟娜，刘文远. MapReduce 框架下基于范围的空间文本相似连接[J]. 小型微型计算机系统，2015，36(10)：2245-2251.

[9] ZHANG M. An Approach of Near-duplicate Detection of Mass Data Based on MapReduce[J]. Research and Exploration In Laboratory, 2014, 33(9): 132-136. (in Chinese)
张敏. 海量数据的 MapReduce 相似度检测[J]. 实验室研究与探索，2014，33(9)：132-136.

[10] LEVENSHTEIN V. Binary codes capable of correcting deletions, insertions, and reversals [J]. Soviet Physics Doklady, 1966, 10(8): 707-710.

[11] DENG D, LI G L, HAO S, et al. MassJoin: A MapReduce-based Method for Scalable String Similarity Joins[C]// Proceedings of IEEE 30th International Conference on Data Engineering (ICDE). New York: IEEE, 2014: 340-351.

[12] WANG J, LI G, FENG J. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints[J]. PVLDB, 2010, 3 (1): 1219-1230.

[13] BAYARDO R J, MA Y, SRIKANT R. Scaling up all pairs similarity search[C]// Proceedings of 16th International World Wide Web Conference(WWW 2007). Banff, AB: ACM, 2007: 131-140.

(下转第 32 页)

参 考 文 献

[1] Top500. org. Top 500 supercomputer site [EB/OL]. [2015-12-09]. http://www. top500. org.

[2] 王恩东,张清,沈铂,等. MIC 高性能计算编程指南[M]. 北京:中国水利水电出版社,2012:15-21.

[3] ZHUKOV V T, KRANSON M M, NOVIKOVA ND, et al. Multigrid effectiveness on modern computing architectures[J]. Programming and Computer Software, 2015, 42(1): 14-22.

[4] REINDERS J, JEFFERS J. High Performance Parallelism Pearls, Multicore and Many-core Programming [M]. Waltham, MA, USA: Morgan Kaufmann, 2014: 377-396.

[5] ITU L M, SUCIU C, MOLDOVEANU F, et al. GPU Optimized Computation of Stencil Based Algorithms[C]// 2011 RoEduNet International Conference 10th Edition: Networking in Education and Reasearch. 2011: 1-6.

[6] WANG G B, YANG X J, ZHANG Y, et al. Program Optimization of Stencil Based Application on the GPU-accelerated System [C]// 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications. 2009: 219-225.

[7] HERNANDEZ M, CEBRIAN J M, CECILIA J M, et al. Evaluating 3-D Stencil codes on Intel Xeon Phi: Limitations and Tradeoffs[C]// XXVI Jornadas De Paralelismo. 2015: 441-444.

[8] KOMATITSCH D, ERLEBACHER G, GODDEKE C, et al. High-order finite element seismic wave propagation modeling with MPI on a large GPU cluster[J]. Journal of Computational physics, 2010, 229(20): 7692-7714.

[9] FANG J B, SIPS H, ZHANG L L, et al. Test- driving Intel Xeon Phi[C]// 5th ACM/SPEC International Conference on Performance Engineering. 2014: 137-148.

[10] LIANG X, WANG Z Y, LIU G H, et al. Numerical simulation of elastic wave based on the staggered grid finite difference method [C]// 2011 International Conference on Consumer Electronics, Communications and Networks. 2011: 3283-3286.

[11] WANG T, LI L G, ZHANGY, et al. Pseudo-spectral method for modeling elastic wave propagation in isotropic medium[C]// 12th International Conference on Signal Processing. 2014: 58-62.

[12] WOSKOBOYNIKOVA G M. Seismic wave propagation in fractured media[C]// Third International Forum on Strategic Technologies. 2008: 307-309.

[13] LIU H W, LI B, LIU H, et al. The Algorithm of high order finite difference pre-stack reverse time migration and GPU implementation [J]. Chinese Journal of Geophysics, 2010, 53(7): 1725-1733. (in Chinese)
刘红伟,李博,刘洪,等. 地震叠前逆时偏移高阶有限差分算法及 GPU 实现[J]. 地球物理学报,2010,53(7):1725-1733.

[14] PERAZA J, TIWARI A, LAURENZANO M, et al. Understanding the performance of stencil computations on Intel's Xeon Phi [C]// IEEE International Conference on Cluster Computing. 2013: 1-5.

[15] SHAN Y, WU J P, WANG Z H. Hierarchial parallel programming model and parallelization and optimization techniques based on SMP cluster[J]. Appolication Research of Computers, 2006, 23(10): 254-256. (in Chinese)
单莹,吴建平,王正华. 基于 SMP 集群的多层次并行编程模型与并行优化技术[J]. 计算机应用研究,2006,23(10):254-256.

[16] SATO M. OpenMP: Parallel Programming API for shared memory multiprocessors and on-chip multiprocessors[C]// 15th International Symposium on System Synthesis. 2002: 109-111.

[17] JEFFERS J, REINDERS J. Intel Xeon Phi coprocessor high-performance programming [M]. 陈建,周珊,李慧等,译. 北京:人民邮电出版社,2014:92-96.

[18] STROUT M M, LUPORINI F, KRIEGER C D, et al. Generalizing Run-time Tiling with the Loop Chain Abstraction[C]// IEEE 28th International Parallel and Distributed Processing Symposium. 2014: 1136-1145.

(上接第 25 页)

[14] XIAO C, WANG W, LIN X, et al. Efficient similarity joins for near-duplicate detection[J]. ACM Trans. Database Syst., 2011, 36(3): 1-41.

[15] XIAO C, WANG W, LIN X. Ed-join: an efficient algorithm for similarity joins with edit distance constraints[J]. Proceedings of the Vldb Endowment, 2008, 1(1): 933-944.

[16] WANG W, QIN J, XIAO C, et al. Vchunkjoin: An efficient algorithm for edit similarity joins [J]. IEEE Trans. Knowl. Data Eng., 2013, 25(8): 1916-1929.

[17] ARASU A, GANTI V, KAUSHI K R. Efficient exact set-similarity joins[C]// Proceedings of the 32nd International Conference on Very Large Data Bases. Seoul, Korea: VLDB Endowment, 2006: 918-929.

[18] LI G, DONG D, WANG J, et al. PASS-JOIN: A Partition-based Method for Similarity Joins[J]. Proceedings of the Vldb Endowment, 2011, 5(3): 253-264.

[19] VERNICA R, CAREY M J, LI C. Efficient parallel set-similarity joins using MapReduce[C]// Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indianapolis, IN, USA: ACM, 2010: 495-506.

[20] METWALLY A, FALOUTSOS C. V-SMART-Join: A Scalable MapReduce Framework for All-Pair Similarity Joins of Multisets and Vectors [J]. Proceedings of the Vldb Endowment, 2012, 5(8): 704-715.

[21] AFRATI F N, SARMA A D, MENESTRINA D, et al. Fuzzy Joins Using MapReduce[C]// Proceedings of IEEE 28th International Conference on Data Engineering. Washington, DC: IEEE, 2012: 498-509.

[22] LIN C, YU H, WENG W, et al. Large-Scale Similarity Join with Edit-Distance Constraints[C]// Proceedings of 19th International Conference on Database Systems for Advanced Applications (DASFAA 2014). Bali, Indonesia: Springer International Publishing, 2014: 328-342.