

一种基于 MapReduce 的局部相似自连接算法

王晓霞,孙德才

(渤海大学 信息科学与技术学院,辽宁 锦州 121013)

摘要:局部相似自连接能在给定的单个数据集中快速找到所有满足相似要求的记录对,它在数据清洗、基因序列比对和剽窃检测等领域都有广泛的应用。为研究基于单个字符串集的并行自连接算法,提出了一种基于 MapReduce 框架的自连接算法,解决了局部相似自连接的定位问题。该算法采用了过滤验证二阶段模式;在过滤阶段,采用无关对过滤和冗余对过滤抛弃了大量的无效字符串对;在验证阶段,通过生成小编号串内容保留项解决了字符串编号和内容的快速配对问题。实验结果显示,该算法在大数据集上的自连接速度一直快于当前的优秀算法 LS-Join,同时非常适合动态编辑距离参数环境下的局部相似自连接操作。实验结果也证明,该算法中提出的相关技术有效地提高了局部相似自连接的速度。

关键词:相似连接;自连接;MapReduce;数据清洗;大数据

中图分类号:TP391

文献标识码:A

文章编号:1673-629X(2020)02-0088-06

doi:10.3969/j.issn.1673-629X.2020.02.018

A MapReduce-based Local Similarity Self-join Algorithm

WANG Xiao-xia, SUN De-cai

(School of Information Science and Technology, Bohai University, Jinzhou 121013, China)

Abstract: Local similarity self-join can find all local similar pairs from a given set quickly, which is widely used in many areas, such as data cleaning, gene sequence alignment, near duplicate detection and so on. In order to study the parallel self-join algorithm based on single string set, a self-join algorithm based on MapReduce framework is proposed to solve the locating problem of local similarity self-join. Filter-verify framework is employed in this algorithm. In filter stage, a lot of dissimilar string pairs are discarded by using the techniques of irrelevant-pair filter and redundant-pair filter. In verify stage, the technique of generating reserved terms is adopted to solve the problem of matching string contents with IDs quickly. Experiment shows that the proposed algorithm outperforms the current excellent algorithm LS-Join on big dataset and performs well on condition of dynamic parameter of edit distance. It also demonstrates that the performance of local similarity self-join is improved by using the techniques of the proposed algorithm.

Key words: similarity join; self-join; MapReduce; data cleaning; big data

0 引言

随着各行各业数据量的飞速增长,大数据的存储、处理、管理和分析等领域已成为当前研究的热点。在大数据处理中,数据清洗的目的是删除冗余信息、纠正存在的错误等。相似连接(similarity join)^[1-4]能在给定的数据集中快速找出所有满足相似要求的记录对,是数据清洗中去除相似信息的常用方法。相似连接在基因序列比对、剽窃检测和信息检索等领域也有广泛的应用。

相似连接分为全局连接和局部连接。全局连接要求记录对的整体相似,而局部连接则要求记录对满足

局部相似要求即可。相似连接又根据参与连接的数据源数分为自连接和多源连接。当数据源为单个时称为自连接,自连接找出的相似记录对都来源于同一个数据集。而当数据源为两个及以上时称为多源连接,多源连接的相似记录对分别来源于不同的数据集。在相似连接中,衡量一个记录对的相似程度的方法主要包括编辑距离^[5-6]、海明距离、Jaccard、Cosine 和 Dice 等。编辑距离是指把一个字符串经过插入、修改或删除三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。文中用编辑距离衡量字符串对的相似度,因为编辑距离不仅对噪声鲁棒性强,还能体现出两个

收稿日期:2019-04-08

修回日期:2019-08-12

网络出版时间:2019-11-07

基金项目:教育部人文社会科学研究青年基金项目(15YJC870021);国家自然科学基金青年基金项目(61602056);辽宁省自然科学基金(20170540015);辽宁省社会科学基金(L18AXW001);辽宁省教育科学研究项目(L2015010)

作者简介:王晓霞(1977-),女,硕士,讲师,研究方向为大数据、相似连接、入侵检测等。

网络出版地址:<http://kns.cnki.net/kcms/detail/61.1450.TP.20191107.0912.060.html>

字符串间字符顺序的差别^[6]。

研究相似连接算法的主要目的是加快相似连接的速度,尤其是基于大数据集的相似连接。当前的相似连接算法主要有内存算法和并行算法。内存算法由于运行过程中允许共享大量信息(如索引等),所以只能运行在单台机器上,如 PassJoin^[7]、K-Join^[8]和 LS-Join^[9]等。并行算法的设计目的是实现集群多任务并行计算,但并行算法也有共享信息困难的问题。MapReduce 框架是 Google 提出的一种高效的分布式编程框架,在大数据处理中应用广泛。近年来,基于 MapReduce 框架的相似连接并行算法^[6,10-15]的研究也得到了众多学者的关注,如 V-SMART-Join^[10]、PassJoinKMR^[6]、MassJoin^[11]、SAX^[14]、FS-Join^[15]等。

文中研究的主要内容是基于单个字符串集的局部相似自连接并行算法。目前,LS-Join 算法是首个基于两个数据集的局部相似连接内存算法,虽然文中也提出了改进的多线程算法,但仍无法实现集群多节点的并行计算。另外,在当前的相关文献中也尚未发现局部相似自连接算法的相关研究。对此,文中提出一种新的基于 MapReduce 框架的并行连接算法,并拟解决局部相似自连接的定位问题。

1 局部相似自连接及新过滤方案

1.1 局部相似自连接的问题定义

给定一个数据集,局部相似自连接将找出集合中所有存在局部相似的记录对。这里先给出相关问题的定义:

定义 1(字符串 $l-\tau$ 局部相似问题):给定一个窗口长度 l 、一个编辑距离参数 τ 和两个字符串 s_i, s_j 。如 r_i, s_j 中存在子串对满足 $\text{ed}(s_i^p, s_j^q) \leq \tau, |s_i^p| \geq l, |s_j^q| \geq l$, 其中 s_i^p 是 s_i 中的一个子串, s_j^q 是 s_j 中的一个子串,则称 $\langle s_i, s_j \rangle$ 为一个 $l-\tau$ 局部相似对。

定义 2(局部相似自连接的定位问题):给定一个窗口长度 l 、一个编辑距离参数 τ 和一个字符串集 S 。局部相似自连接的定位问题是从字符串集 S 中找出所有存在 $l-\tau$ 局部相似的串对 $\langle s_i, s_j \rangle, s_i \in S, s_j \in S, i \neq j$, 并同时找出该串对中最长的局部相似子串位置。

表 1 例子字符串集

String setS
1#GCACTACTGAACG
2#AGACAGCRR
3#GTACTCAACGATAGC

如给定一个字符串集 S , 如表 1 所示。数据集中每行是一个字符串的信息, 其中‘#’号前面的是字符串编号, 而后面的是字符串内容。如 $l=7, \tau=1$, 则

$\langle s_1, s_3 \rangle$ 是一个 $l-\tau$ 局部相似对, 它们间如最长子串对为 $\text{ed}(s_1[4, 12], s_3[1, 9])=1$ 。

1.2 局部相似自连接过滤方案

局部相似自连接的输入包括一个给定的数据集、一个窗口长度和一个编辑距离参数。LS-Join 算法^[9]是一种基于两个数据集的多源局部相似连接算法。LS-Join 算法先读取第一个字符串集并建立一个倒排索引。然后把第二个字符串集中的字符串拆分子串, 并在倒排索引中检索子串并生成候选串对。最后, 提出了一种基于双向扩展的局部验证方法来验证候选对。该算法也能进行自连接运算, 只需输入的两数据集相同即可。但如此连接后, 结果集中存在两种冗余记录对, 即自身冗余对和正反冗余对。自身冗余对为字符串和本身组成的串对, 该串对中因两字符串完全一样, 所以一定存在于结果集中, 即 $\langle s_i, s_j \rangle, s_i \in S, s_j \in S, i=j$ 。正反冗余串对是两个不同编号的字符串组成的串对, 但因分先后顺序而形成的冗余, 如 $\langle s_i, s_j \rangle, s_i \in S, s_j \in S, i \neq j$ 和 $\langle s_j, s_i \rangle, s_i \in S, s_j \in S, i \neq j$ 。这些冗余串对不仅增加了连接算法计算的时间, 还增加了结果集的数量。

因给定的数据集中含有大量的字符串, 因此也存在着海量的字符串对。为避免枚举所有串对, 文中算法采用了过滤验证二阶段模式。为进行快速过滤和去除冗余对, 该算法采用了基于分割子串的过滤方案。

定理 1(无关对过滤定理): 给定两个字符串 s_i, s_j , 一个编辑距离参数 τ 和一个窗口长度 l 。把串 s_i 分割成 $\lfloor |s_i|/q \rfloor$ 个连续但不重叠的长度为 q 的子串(称为 Q-sample), 其中 $q = \lfloor (l+1)/(\tau+2) \rfloor, q \geq 1$ 。此时如字符串对 $\langle s_i, s_j \rangle$ 为 $l-\tau$ 局部相似对, 则串 s_j 中包含至少一个串 s_i 的 Q-sample。

证明: 如字符串对 $\langle s_i, s_j \rangle$ 为 $l-\tau$ 局部相似对, 则一定存在 $\text{ed}(s_i^p, s_j^q) \leq \tau$, 其中 s_i^p 是 s_i 中一个子串($|s_i^p| \geq l$) 和 s_j^q 是 s_j 中一个子串($|s_j^q| \geq l$)。因为 $q = \lfloor (l+1)/(\tau+2) \rfloor$, 所以 s_i 中任何一个长度不小于 l 的窗口(包括 s_i^p) 内至少含有 $\tau+1$ 个完整的 Q-sample。因 $\text{ed}(s_i^p, s_j^q) \leq \tau$ 和编辑距离原理, 则可通过不大于 τ 次编辑操作(插入、修改和删除)把 s_i^p 转换成 s_j^q 。因串 s_i 分割得到的 Q-sample 是连续但不重叠的, 所以根据鸽巢原理 τ 次编辑操作最多破坏 τ 个 Q-sample。因此, 至少存在一个 Q-sample 未被破坏, 定理 1 成立。

定理 2(冗余对过滤定理): 给定一个字符串集 S , 一个编辑距离参数 τ 和一个窗口长度 l 。字符串集 S 中有两个字符串 s_i, s_j , 其中 i 为字符串 s_i 在集合 S 中的编号和 j 为 s_j 的编号, 则字符串对集 $G = \{ \langle s_i, s_j \rangle, \langle s_j, s_i \rangle \}$ 中不存在 $l-\tau$ 局部相似对。

$i < j$ 中一定不存在冗余串对。

证明:在局部相似自连接中,有两种冗余串对,即自身串对和正反串对。条件 $i < j$ 使得集合 G 中没有自身冗余对,同时也只包含正反串对中一个前小后大的串对。因此,定理2成立。

2 基于 MapReduce 框架的局部相似自连接算法

局部相似自连接的输入包括:一个字符串集 S , 一个编辑距离参数 τ 和一个窗口长度 l 。为实现并行计算,文中的 MLSSJ 算法采用了分布式编程框架 MapReduce,共设计了三个阶段,即过滤阶段、验证阶段1和验证阶段2。

2.1 过滤阶段

为避免枚举所有可能的字符串对,该算法设计了一个 MapReduce 任务实现无关对和冗余对的快速过滤方案。定理1是一个无关对过滤条件,使用定理1能抛弃那些不共享 Q-sample 的字符串对。定理2是一个冗余对过滤条件,使用定理2能抛弃影响连接性能的冗余对。为使用定理1和定理2进行快速过滤,MLSSJ 算法在过滤阶段先对输入的字符串集 S 中所有字符串进行子串分割,然后再进行过滤。过滤阶段的 MapReduce 任务包括三个过程,即 Map、Shuffle 和 Reduce。

(1)Map 过程:基于 MapReduce 框架设计的程序执行中并行交替运行着众多的 map 任务,每次 map 任务的输入是一个 key-value 对 $\langle \text{sn}, \text{split} \rangle$, 其中 sn 是分片的编号,而 split 是输入字符串集 S 的一行内容。例如表1中的例子集合,一次 map 的 split 就是一行内容,即一个字符串的编号和内容。为进行局部相似自连接的前期处理,该算法先提取字符串的编号(记为 sid)和字符串内容(记为 s),然后针对字符串 s 分别生成索引子串和匹配子串,具体过程如下:

生成索引子串:为使得其他字符串能够匹配到该字符串,该算法把字符串 s 分割成 $\lfloor |s|/q \rfloor$ 个连续但不重叠的长度为 $q = \lfloor (l+1)/(\tau+2) \rfloor, q \geq 1$ 的 Q-sample,如存在长度不足 q 的子串则直接抛弃。设 $s[p_i^i, p_i^i + q - 1]$ 表示第 i 个 Q-sample,且 p_i^i 为起点,且 $1 \leq i \leq \lfloor |s|/q \rfloor, p_i^1 = 0, p_i^i = (i-1)q$ 。最后输出这些 Q-sample 对应的索引子串位置信息的 key-value 对,即 $\langle s[p_i^i, p_i^i + q - 1], (I', \text{sid}_i^i, p_i^i) \rangle$, 其中 sid_i^i 为该字符串的编号 sid, I' 符号代表该项为一个索引子串, p_i^i 为该索引子串的偏移量。

生成匹配子串:为使用子串去匹配其他字符串的索引子串,该算法把该字符串 s 拆分成连续重叠且长

度为 q 的子串(相邻 q-gram 重叠 $q-1$ 个字符,这里称为 q-gram, q-gram 因可重叠而不同于 Q-sample), 获得 $s-q+1$ 个 q-gram。设 $s[p_M^j, p_M^j + q - 1]$ 表示第 j 个 q-gram, 其中 p_M^j 为起点,且 $1 \leq j \leq |s| - q + 1, p_M^1 = 0, p_M^j = j - 1$ 。最后输出这些 q-gram 对应的匹配子串位置信息的 key-value 对,即 $\langle s[p_M^j, p_M^j + q - 1], (M', \text{sid}_M^j, p_M^j) \rangle$, 其中 sid_M^j 为该字符串的编号 sid, M' 符号代表该项为一个匹配子串, p_M^j 为该匹配子串的偏移量。

(2)Shuffle 过程:在 MapReduce 框架中,shuffle 过程将 map 过程产生的所有 key-value 对按 key 值(索引子串和匹配子串)进行混淆、排序,并把具有相同 key 的 key-value 对送到同一 reduce 节点上。

(3)Reduce 过程:同样基于 MapReduce 框架设计的程序执行中也并行交替运行着众多 reduce 任务,每次 reduce 任务将处理 shuffle 结果中的一个 key-value 对,即 $\langle \text{sig}, \text{list}((I', \text{sid}_i^i, p_i^i)/(M', \text{sid}_M^j, p_M^j)) \rangle$, 其中 sig 是子串内容,后面是该 sig 对应的所有索引子串和匹配子串的位置信息列表。为便于处理,该算法先根据子串类型把列表划分成两个列表,即把类型为 I' 的索引子串项 $[\text{sid}_i^i, p_i^i]$ 存储到 Ilist 列表中,把类型为 M' 的匹配子串项 $[\text{sid}_M^j, p_M^j]$ 存储到 Mlist 列表中。划分完毕后根据定理1可知,如 Ilist 或 Mlist 为空列表,则可知该 sig 中不存在候选串对。否则根据定理1可知, Ilist 任意项与 Mlist 任意项的组合可能就是一个存在局部相似的候选对。为获得所有候选对,这里先循环列表 Mlist, 设 $\text{Mlist}[j]$ 为第 j 个元素。为枚举 $\text{Mlist}[j]$ 与 Ilist 间的所有候选对,需完整遍历一次 Ilist 列表。如 $\langle \text{Ilist}[i], \text{Mlist}[j] \rangle$ 是一个候选对, 设 $\text{Ilist}[i] = (\text{sid}_i^i, p_i^i), \text{Mlist}[j] = (\text{sid}_M^j, p_M^j)$ 。根据定理2可知,如 $\text{sid}_i^i < \text{sid}_M^j$, 则把 $\text{sid}_i^i; p_i^i; p_M^j$ 添加到一个新 list 中,这里记 sid_M^j 为大编号串, sid_i^i 为小编号串。如此处理直到循环完 Ilist 后,生成一个 $\text{Mlist}[j]$ 的候选对集 key-value 对,即 $\langle \text{sid}_M^j, \text{list}(\text{sid}_i^i; p_i^i; p_M^j) \rangle$ 。如此循环处理 Mlist, 生成多个 key-value 对。直到处理完 Mlist 列表后,该次 reduce 任务结束。

2.2 验证阶段1

过滤阶段输出结果由大量的 key-value 对构成,其中每个 key-value 对都是一个候选对集,即一个 key-value 对包含了某个字符串与集 S 中所有编号小于该串的候选对。而验证阶段的任务是从这些候选对集中找出真正含有局部相似的串对,并定位最长相似子串的位置。但现在因候选对集只有串编号而没有串内容,所以算法无法验证。为实现候选对集字符串内容的快速读取配对和验证候选对,该算法把验证阶段设

计成了两个阶段,即验证阶段1和验证阶段2。

验证阶段1的主要目的是读取集 S 的字符串内容,并初步进行字符串编号和串内容的配对。它的输入包括集 S 和过滤阶段的输出结果。验证阶段1的MapReduce任务也包含Map、Shuffle和Reduce三个过程。

(1)Map过程:因为验证阶段的输入源有两个,所以每个map任务根据数据来源的不同进行不同的处理。如读入的key-value对来源于集 S ,则是一个字符串的信息;首先从split中获取字符串编号sid和字符串内容 s ,然后直接输出key-value对 $\langle \text{sid}, \#s \rangle$,其中 $\#$ 标识该项为字符串内容。如读取的key-value对来源于过滤阶段的输出结果,则先直接原样输出一个 $\langle \text{sid}_M^i, \text{list}(\text{sid}_I^i; p_I^i; p_M^i) \rangle$ 。如此能够实现大编号 sid_M^i 和串集 S 生成的 $\langle \text{sid}, \#s \rangle$ 串内容的配对,即 $\text{sid}_M^i = \text{sid}$ 时配对成功。但 $\text{list}(\text{sid}_I^i; p_I^i; p_M^i)$ 中的小编号 sid_I^i 无法与 $\langle \text{sid}, \#s \rangle$ 中的串内容实现配对,因此需要在map中保留这些将来要用到的串内容,而既不是大编号 sid_M^i 也不是小编号 sid_I^i 的 $\langle \text{sid}, \#s \rangle$ 串内容将会在reduce中抛弃。因此算法将循环处理 $\text{list}(\text{sid}_I^i; p_I^i; p_M^i)$ 中的每个项,并把每个项中的小编号 sid_I^i 添加到一个非重复的集合 H 中(sid_I^i 去重)。最后遍历集合 H 为每个小编号 sid_I^i 输出一个保留项key-value对 $\langle \text{sid}_I^i, I \rangle$ 。

(2)Shuffle过程:对map的输出按sid(sid_M^i 和 sid_I^i)进行混淆、排序,并将结果作为reduce的输入。

(3)Reduce过程:每次reduce过程的输入是一个key-value对,即 $\langle \text{sid}, \text{list}(I / \text{list}(\text{sid}_I^i; p_I^i; p_M^i) / (\#s)) \rangle$,该sid对中可能包含候选对集、串内容和小编号保留项。该算法先循环遍历列表 $\text{list}(I / \text{list}(\text{sid}_I^i; p_I^i; p_M^i) / (\#s))$ 的所有项。如访问项的第一个字符是 $\#$,则该项是串sid的串内容,保存到 s 中备用;如访问项的第一个字符是 I ,则代表该串内容是需要保留的小编号串,并记 $F = \text{true}$;否则访问项是 $\text{list}(\text{sid}_I^i; p_I^i; p_M^i)$,需循环处理列表 $\text{list}(\text{sid}_I^i; p_I^i; p_M^i)$ 中每个 $\text{sid}_I^i; p_I^i; p_M^i$,即先提取小编号 sid_I^i 和匹配位置对 $p_I^i; p_M^i$,再把 $p_I^i; p_M^i$ 添加到一个哈希集 $\text{HS}[\text{sid}_I^i]$ 中。如此处理直到遍历完 $\text{list}(I / \text{list}(\text{sid}_I^i; p_I^i; p_M^i) / (\#s))$ 中所有的项。此时如 $F = \text{true}$,则该sid的串内容是需要保留的,输出一个key-value对 $\langle \text{sid}_I^i, \#s_i^i \rangle$,其中 $\text{sid}_I^i = \text{sid}$, $s_i^i = s$,否则不输出。最后,遍历哈希集合 HS 中每个小编号 sid_I^i 同时提取其匹配位置列表,并以 $\text{sid}_I^i. \text{list}(p_I^i; p_M^i)$ 格式作为一项添加到一个新列表Clist中,其中 $\#$ 作为分隔小编号 sid_I^i 和 $\text{list}(p_I^i; p_M^i)$ 的标志。遍历完成后构建一个key-value对 $\langle \text{sid}_M^i \# s_M^i, \text{Clist}(\text{sid}_I^i. \text{list}(p_I^i;$

$p_M^i)) \rangle$ 并输出,其中 $s_M^i = s$ 。

2.3 验证阶段2

验证阶段1结束后实现了大编号串编号和串内容的匹配,也给出了每个大编号串对应的候选对集,还输出了需要保留的小编号串内容。但此时仍无法进行候选对的验证工作,因为还缺少小编号串的串内容。验证阶段2的任务是实现小编号串编号和串内容的配对,同时进行最终的验证定位工作。验证阶段2的输入是验证阶段1输出结果。验证阶段2依然包含Map、Shuffle和Reduce三个过程。

(1)Map过程:每个map任务根据输入数据的内容不同而进行不同的处理。首先获取输入key-value对中的key,如key值中不含有 $\#$,则输入的数据是要保留的小编号串内容,这里直接原样输出,即 $\langle \text{sid}_I^i, \#s_i^i \rangle$ 。否则,输入是 $\langle \text{sid}_M^i \# s_M^i, \text{Clist}(\text{sid}_I^i. \text{list}(p_I^i; p_M^i)) \rangle$ 。该算法中依次处理Clist中的每个项。针对其中一个项 $\text{sid}_I^i. \text{list}(p_I^i; p_M^i)$,先根据 $\#$ 位置提取出 sid_I^i 和 $\text{list}(p_I^i; p_M^i)$,然后输出一个key-value对,即 $\langle \text{sid}_I^i, \text{list}(p_I^i; p_M^i) \# \text{sid}_M^i \# s_M^i \rangle$,其中 $\#$ 是一个分隔匹配列表和大编号串信息的一个分隔符。如此处理,算法针对Clist中每个项都输出了一个key-value对。

(2)Shuffle过程:对map的输出按 sid_I^i 进行混淆、排序,并将结果作为reduce的输入。

(3)Reduce过程:每个reduce任务的输入是 $\langle \text{sid}_I^i, \text{list}((\text{list}(p_I^i; p_M^i) \# \text{sid}_M^i \# s_M^i) / (\#s_i^i)) \rangle$ 。该key-value对中既包含串 sid_I^i 对应的所有候选对信息,还含有该小编号串的内容。为获取该小编号串的内容,算法先循环处理列表的所有项。如访问项的第一个字符是 $\#$,则该项是串 sid_I^i 的串内容,提取并保存到 s_i^i 中备用。否则访问项是 $\text{list}(p_I^i; p_M^i) \# \text{sid}_M^i \# s_M^i$,则把它保存到一个新列表Dlist中备用。循环处理完所有项后,大编号和小编号串的串内容都已获取到,此时已具备验证定位的条件。为验证小编号串 sid_I^i 对应的每个候选对,需循环处理列表Dlist中每个项。针对其中一项 $\text{list}(p_I^i; p_M^i) \# \text{sid}_M^i \# s_M^i$,算法先根据 $\#$ 和 $\#$ 位置提取出串对间的匹配集 $L = \text{list}(p_I^i; p_M^i)$ 、大编号串的编号 sid_M^i 以及大编号串的内容 s_M^i 。为降低验证候选对的时间消耗,该算法采用优化后的双向扩展验证方法^[9]来验证每个候选对 $\langle \text{sid}_I^i, \text{sid}_M^i \rangle$,如该串对是存在 $l - \tau$ 局部相似,则获取评分最高的子串对 $\langle s_i^p, s_j^q \rangle$,其中 s_i^p 是 s_i^i 的一个子串, s_j^q 是 s_M^i 的一个子串。最后,输出该真实匹配对的结果key-value对(包括串对内容、最长相似子串位置和子串间编辑距离),即 $\langle \text{ed}(s_i^p, s_j^q), \{(\text{start}(s_i^p), \text{end}(s_i^p)) \text{sid}_I^i \# s_i^i, (\text{start}(s_j^q), \text{end}(s_j^q)) \text{sid}_M^i \# s_M^i\} \rangle$ 。直到处理完Dlist中所有项,

reduce 过程结束。

3 实验

3.1 实验环境

为验证文中算法和相关技术的有效性,实验中基于 MapReduce 框架用 Java 实现了文中的 MLSSJ 算法。算法运行环境为 Hadoop 集群,集群中主节点 1 个,从节点 4 个;硬件配置均为 CPU i5 4590 四核心、16 G 内存和 1 TB 硬盘。实验中还实现了文献[9]中的 LS-Join 算法,这里记 LSJ-S 为单线程算法,记 LSJ-M 为多线程算法(线程数为 4)。实验的数据主要来源于两个数据集,见表 2。DBLP 集是一个计算机类英文文献的集成数据库,实验中只保留了记录中作者和标题两个字段。GBEST 集为 NCBI GenBank 的表达序列标签(Expressed Sequence Tags, <ftp://ftp.ncbi.nlm.nih.gov/>),实验中只保留了序列本身。

表 2 实验数据集信息

	DBLP	GBEST
Size/MB	23.0	15.5
Number of strings	240,000	45,000
Average length	92.0	355.6
Alphabet size	95	15

3.2 算法性能对比及结果分析

评价相似连接算法时,时间性能最为重要。LS-Join 算法实验中的最优配置参数 DBLP 集为 $l=50, \tau=5, q=7$, GBEST 集为 $l=100, \tau=7, q=9$ 。MLSSJ 算法的配置参数除不需要 q 值外其他与 LS-Join 算法相同。首先在实验中对不同大小数据集对各算法性能的影响。实验中分别采用 LSJ-S、LSJ-M 和 MLSSJ 算法在不同大小的字符串集分别进行局部相似自连接运算,并统计了各个算法自连接的时间消耗。随着字符串集字符串数量的逐渐增大,各算法的总连接时间变化如图 1 和图 2 所示。

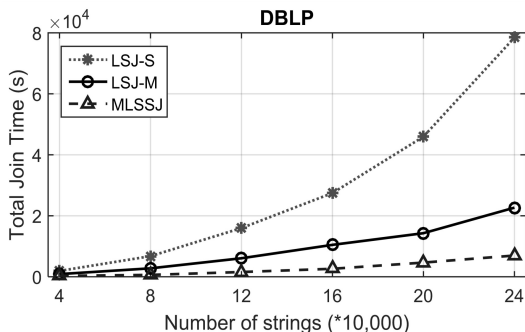


图 1 DBLP 集大小与连接时间

由图 1 和图 2 可知,随着数据集中字符串数量的不断增大,LSJ-M 算法的连接速度一直快于 LSJ-S 算法,由此可见多线程并发技术加快了 LS-Join 算法的

连接速度。MLSSJ 算法的连接速度要明显快于 LS-Join 的 LSJ-S 算法和 LSJ-M 算法,尤其在小字母表且长字符串的数据集(如 GBEST)上新算法性能表现更优。这主要是因为 LS-Join 算法是一个内存算法,虽然采用了多线程技术但也只能运行在一台机器上。而 MLSSJ 算法是一个运行在多节点集群上的并行算法,基于 MapReduce 框架的设计使得它更适合大数据集的局部相似自连接。从图中还可以看出,随着数据集的增大,MLSSJ 算法的连接时间基本呈线性增加。

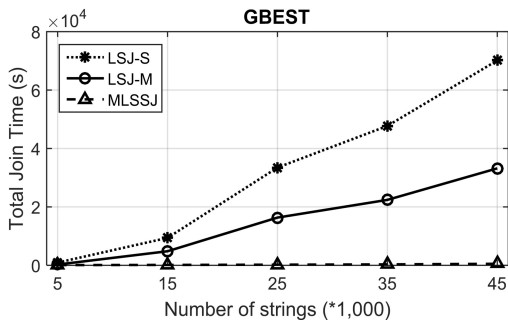


图 2 GBEST 集大小与连接时间

在相似连接中,编辑距离参数是一个非常重要的参数。为对比各个算法对编辑距离参数的敏感度,实验中还分别采用不同的编辑距离参数对 LSJ-M 算法和 MLSSJ 算法在各数据集上进行了局部相似自连接操作。在 DBLP 集上配置参数为 $l=50, \tau=0, 1, 3, 5$ (LSJ-M 中 $q=7$), GBEST 集上为 $l=100, \tau=0, 1, 3, 5, 7$ (LSJ-M 中 $q=9$)。随着编辑距离参数的逐渐增大,各算法的性能表现对比如图 3 和图 4 所示。

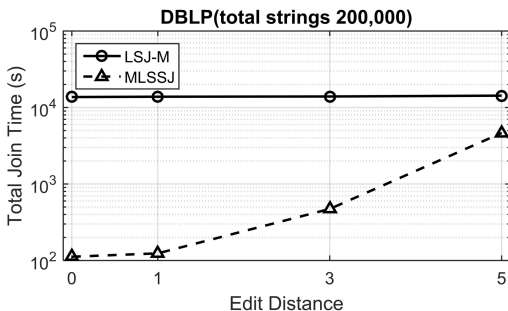


图 3 DBLP 集编辑距离与连接时间

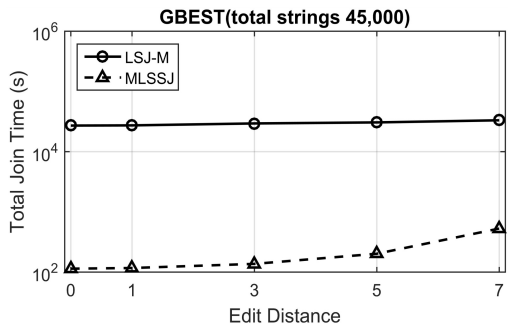


图 4 GBEST 集编辑距离与连接时间

由图 3 和图 4 可知,在不同编辑距离参数下 MLSSJ 算法的速度一直快于 LSJ-M 算法。该实验主

要分析各个算法对编辑距离参数的敏感度。由图 3 和图 4 可知,LSJ-M 算法随着编辑距离参数不断增大而连接时间变化较小,因此该算法对编辑距离参数的敏感度较小。这主要是因为实验中 LSJ-M 算法的索引子串长度 q 值对不同编辑距离参数影响较大,而这里 q 值固定。实际使用中需要对不同数据集进行 q 值实验测优,因此该算法也不够灵活。而 MLSSJ 算法随着编辑距离参数的不断增大,其连接时间也不断增加,因此该算法对编辑距离参数的敏感度较大。这主要是因为 MLSSJ 算法中切分子串长度是根据编辑距离参数动态计算的,即 τ 值越小, q 值越大,分割得到的子串数目越少,处理所需的时间就越短。因此,MLSSJ 算法在编辑距离参数动态变化的局部相似自连接中比 LSJ-M 算法更具优势。

4 结束语

文中主要研究了基于单个字符串集的局部相似自连接算法。先给出了局部相似自连接定位问题的定义,然后通过分析相似自连接中的无关串对和冗余串对问题,总结出了无关串对过滤定理和冗余串对过滤定理。最后提出了一种基于 MapReduce 框架的局部相似自连接并行算法。实验结果表明,该算法有效地提高了局部相似自连接的速度。该算法虽然通过 MapReduce 框架的并行技术加快了局部相似自连接的速度,但仍存在并行节点间数据传输量大和过滤阶段生成的候选对多等问题。下一步将研究更加苛刻的过滤条件,拟通过降低过滤阶段生成的候选对数量来减少验证时间。

参考文献:

[1] YU M, LI G, DENG D, et al. String similarity search and join: a survey[J]. *Frontiers of Computer Science*, 2016, 10(3): 399–417.

[2] CHEN G, YANG K, CHEN L, et al. Metric similarity joins using MapReduce[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 29(3): 656–669.

[3] 王洪亚, 杨利宏, 刘晓强. Top-k 相似连接算法性能优化[J]. *软件学报*, 2016, 27(12): 3051–3066.

[4] PACH R. Large-scale similarity joins with guarantees[C]//

Proceedings of 18th international conference on database theory. Brussels, Belgium: [s. n.], 2015: 15–24.

[5] LEVENSHTAIN V. Binary codes capable of correcting deletions, insertions, and reversals[J]. *Soviet Physics Doklady*, 1965, 163: 845–848.

[6] 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*. Bali, Indonesia: Springer, 2014: 328–342.

[7] LI G, DENG 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.

[8] SHANG Z, LIU Y, LI G, et al. K-join: knowledge-aware similarity join[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2016, 28(12): 3293–3308.

[9] WANG J Y, YANG X C, WANG B, et al. LS-join: local similarity join on string collections[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 29(9): 1928–1942.

[10] 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.

[11] DENG D, LI G L, HAO S, et al. MassJoin: a MapReduce-based method for scalable string similarity joins[C]//*2014 IEEE 30th international conference on data engineering*. Chicago, IL: IEEE, 2014: 340–351.

[12] 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.

[13] VERNICA R, CAREY M J, LI C. Efficient parallel set-similarity joins using MapReduce[C]//*Proceedings of 2010 ACM SIGMOD international conference on management of data*. Indianapolis: ACM, 2010: 495–506.

[14] MA Y, MENG X, WANG S. Parallel similarity joins on massive high-dimensional data using MapReduce[J]. *Concurrency and Computation: Practice and Experience*, 2016, 28(1): 166–183.

[15] RONG C T, LIN C B, SILVA Y N, et al. Fast and scalable distributed set similarity joins for big data analytics[C]//*2017 IEEE 33rd international conference on data engineering (ICDE)*. San Diego, CA: IEEE, 2017: 1059–1070.