一种基于 Q-sample 的局部相似连接并行算法

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 局部相似连接能快速找出数据集间的局部相似记录对,是基因序列比对、剽窃检测和数据清洗等研究领域的基本操作。文中主要研究基于 MapReduce 框架的并行相似连接技术,提出了一种基于 Q-sample 的局部相似连接算法,解决了局部相似连接的定位问题。该算法采用了过滤验证二阶段模式:在过滤阶段,所提算法使用 Q-sample 分割方案拆分字符串集,在不丢失任何匹配的基础上生成了高质量的子串,抛弃了大量的无关字符串对;在验证阶段,所提算法优化了 LS-Join 算法的双向扩展验证方法,通过去除冗余匹配、合并连续匹配和合并非连续匹配等技术提高了算法的验证效率。通过实验对比了不同数据集和编辑距离参数下算法的性能表现,结果显示所提算法在大数据集上的局部相似连接速度快于当前的优秀算法 LS-Join。理论分析和实验结果证明,所提算法的相关技术提高了局部相似的连接性能。

关键词 相似连接,Q-sample,MapReduce,数据清洗,大数据

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/jsjkx.190100240

Q-sample-based Local Similarity Join Parallel Algorithm

WANG Xiao-xia SUN De-cai

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

Abstract Local similarity join can finds all local similar pairs from sets quickly, which is a basic operation in many areas, such as gene sequence alignment, near duplicate detection, data cleaning and so on. This paper focused on designing similarity join parallel algorithm with MapReduce, and proposed a Q-sample-based algorithm to solve the locating problem of local similarity join. The proposed algorithm employs a filter-verify based framework. In filter stage, a Q-sample partition scheme is adopted to generate high-quality signatures without losing any true pairs, and then more dissimilar string pairs are discarded. In verify stage, the LS-Join's backward-forward verification method is improved with the technique of removing redundant match, combining consecutive match and combining non-consecutive match. In the experiments, the performances of the proposed algorithm with different size of datasets or different value of edit distances are scaled. Experimental results show that the proposed algorithm outperforms the current excellent algorithm LS-Join on big dataset. Theoretical analysis and experimental result demonstrate that the performance of local similarity join is improved by using the techniques of the proposed algorithm.

Keywords Similarity join, Q-sample, MapReduce, Data cleaning, Big data

1 引言

随着互联网信息技术的发展,数据呈爆炸式增长,全球已步入大数据时代。同时如何存储、管理和分析这些大数据资源已成为当前研究的热点问题。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要求的相似记录对,是数据清洗和数据集成中的一项基本操作。相似连接分为全局相似连接和局部相似连接两种。全局相似连接要求整个字符串相似,而局部相似连接则要求相似记录对间存在相似的子串即可。局部相似连接在基因序列比对、剽窃检测和数据清洗等方向都有广泛的应用。

在相似连接中,衡量两个记录间相似度的方法主要有两种,一种是基于字符的相似度计算函数,如编辑距离[5-6]和海明距离等,另一种是基于集合的相似度计算函数,如 Jaccard, Cosine 和 Dice 等。编辑距离(Edit Distance)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。

字符串作为一类最常用的数据类型,被广泛应用于各行各业,如网站信息、学术出版、基因序列、信息检索等。本文主要研究基于编辑距离的字符串集局部相似连接算法,用编辑

到稿日期:2019-01-29 返修日期:2019-06-01 本文受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),国家社会科学基金项目(19BTQ028),辽宁省自然科学基金(20170540015),辽宁省社会科学基金(L18AXW001),辽宁省教育厅科学研究项目(L2015010)资助。

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

距离衡量两个字符串间的相似度主要有两方面的优势[6]，一是它能体现出二者字符顺序的差别，二是它对噪声（书写错误等）更加鲁棒。

目前，相似连接算法根据并行性可分为两类，即内存算法和并行算法。内存算法运行于单机上，该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。而并行算法运行在集群上，其优点是实现了多机的并行计算，但并行也产生了多节点间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，在大数据处理中被广泛使用。

本文研究的主要内容是基于两个字符串集的局部相似连接并行算法。LS-Join[9] 是一个局部相似连接的内存算法。该算法首次提出并解决了二字符串集的局部连接问题，尤其在局部相似验证算法的优化方面提出了双向扩展的验证方法。但 LS-Join 算法是一个内存算法，虽然该文献也提出了一种改进的多线程算法，但仍无法实现多节点的并行计算。本文提出了一种新的基于 MapReduce 框架的局部相似连接算法，并优化了 LS-Join 算法的双向扩展验证方法。本文的主要工作如下：

1）提出了一种基于 Q-sample 的局部相似连接并行算法 MQLS-Join，解决了局部相似连接的定位问题。

2）提出了一种基于 Q-sample 的字符串分割方案。使用该方案对字符串集进行分割，不仅能获得高质量的分割子串，还能减少并行计算节点间的数据传输量。

3）提出了去除无效匹配、合并连续匹配和合并非连续匹配区域等技术，优化了 LS-Join 算法的双向扩展验证方法。

2　相关工作

2.1　问题定义

定义 1（局部相似）　给定两个字符串 r_i, s_j，窗口长度 l 和编辑距离参数 τ，设 r_i^p 是 r_i 的一个子串，s_j^q 是 s_j 的一个子串。若 r_i, s_j 满足 $ed(r_i^p, s_j^q) \leqslant \tau, |r_i^p| \geqslant l, |s_j^q| \geqslant l$，则称 $\langle r_i, s_j \rangle$ 为一个局部相似对。

本文主要研究局部相似连接的定位问题，其具体定义[9]如下。

定义 2（局部相似连接的定位问题）　给定两个字符串集 R 和 S，窗口长度 l 和编辑距离参数 τ。局部相似连接的定位问题是从两个集合中找出所有存在局部相似的串对 $\langle r_i, s_j \rangle$，$r_i \in R, s_j \in S$，并同时定位出该串对中最长的局部相似子串位置，即取 $\max\{v\}, v = |r_i^p| - ed(r_i^p, s_j^q)$，且当 $\max\{v\}$ 值相同时取 $\min\{ed(r_i^p, s_j^q)\}$ 值最小的子串。

设 $r_i^p = r_i[p_r^b, p_r^e], s_j^q = s_j[p_s^b, p_s^e]$，其中 $r_i[p_r^b, p_r^e]$ 是 r_i 的一个从第 p_r^b 个字符开始到第 p_r^e 个字符结束的子串（从 0 开始），$s_j[p_s^b, p_s^e]$ 是 s_j 的一个子串。例如，给定两个集合，如表 1 所列，每行中 # 符号前面的数字为字符串编号，后面的内容为字符串的内容。如给定 $l=7, \tau=1$，则 $\langle r_2, s_2 \rangle$ 为一个局部相似对，即两个字符串间存在长度不小于 7 的编辑距离不大于 1 的子串对。定位问题还需找出它们当中最长匹配子串的位置，如最长子串对 $ed(r_2[9,17], s_2[4,12]) = 1$，即 2 # ATCATGCAC[TACTGAACG] 和 2 # GCAG[TACTCAACG]ATAGC。

表 1　例子集合

Table 1　Example of string sets

集合 R	集合 S
1 # AGACAGCRRAARCDRAGG	1 # DCCADGGCRAARDCRCDD
2 # ATCATGCACTACTGAACG	2 # GCAGTACTCAACGATAGC

2.2　相关工作

绝大多数相似连接的内存算法都采用了过滤验证的二阶段模式。在过滤阶段用过滤条件快速抛弃不可能相似的无关对，然后得到候选对集；在验证阶段则采用验证算法处理候选对集以确定最终的连接结果。AllPair[10] 是一个基于 q-gram 的前缀过滤连接算法。它先把字符串分割成 q-gram 并取前 $q\tau+1$ 个 q-gram 按预定顺序排序，然后去除没有公共 q-gram 的串对，最后验证剩余的候选对。EDJoin[11] 采用了位置过滤和内容过滤，去除了部分无效的 q-gram 匹配，减少了需要验证的候选对。PPJoin[12] 则通过位置过滤改进了前缀过滤方法，提高了算法的过滤效率。PassJoin[13] 是一个基于编辑距离的相似连接算法，它在 2013 年 EDBT 组织的相似连接国际大赛中获得了冠军，它的重大改进是连接时生成了高质量的分割子串。VChunkJoin[14] 把字符串分割成不同长度的子串 qchunks，并采用前缀过滤方法进行相似连接。K-Join[15] 是一个基于知识感知的相似连接算法，它用知识划分来衡量字符串间相似度的新度量标准。以上都是全局相似连接算法的研究，关于局部相似连接算法，Wang 等[9] 首次提出了局部相似连接的问题，并设计了一个称为 LS-Join 的算法，解决了局部相似的检测和定位问题。他们还提出了一种双向扩展的局部相似验证方法。

目前，已有许多基于 MapReduce 框架的全局相似连接并行算法。Vernica 等[16] 实现了一种基于 MapReduce 框架的集合相似连接算法。该算法使用了前缀过滤技术，在 map 阶段用字符串前缀项作为 key，用字符串本身作为 value；在 reduce 阶段，共享相同前缀项的字符串对即为候选对。MetWally 等[17] 提出了一个二阶段的 V-SMART-Join 算法，该算法也采用单个项作为 key。Afrati 等[18] 提出了多个相似连接算法，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[6] 对内存算法 PassJoin 进行改进升级，实现了一种名为 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[7] 也基于 PassJoin 算法在 MapReduce 上实现了一种 MassJoin 算法。该算法通过合并相同 key-value 对的方法减少了数据的传输量，但并未降低算法的过滤能力，其不仅支持字符串连接，还被扩展到集合连接。Ma 等[19] 针对高维数据提出了一种基于 MapReduce 的高阶数据相似连接算法。Rong 等[20] 提出了一种用于集合连接的 FS-Join 算法。该算法设计了 3 种高效的用于无关对过滤的过滤器。这些过滤器在过滤时无须计算串对的相似度，减少了过滤的时间消耗。目前，暂未发现基于 MapReduce 的局部相似连接的相关研究。

3　Q-sample 分割方案及局部验证算法优化

在局部相似连接的定位问题中，两个集合间存在大量的可能串对。为了避免枚举所有串对，所提算法采用了过滤验证二阶段模式。本节先介绍 Q-sample 分割方案，然后引出用

于过滤的 Q-sample 过滤定理，最后介绍对 LS-Join 双向扩展验证方法[9]的优化技术。

3.1 Q-sample 分割方案

本文给定一个字符串 r，一个编辑距离参数 τ 和一个窗口长度 l。为减少产生的分割子串和支持并行计算，本文进行如下分割。

定义 3 把字符串 r 分割成$\lceil |r|/q \rceil$个连续但不重叠的子串，其中 q 为子串的长度，$q=\lfloor (l+1)/(\tau+2) \rfloor$，且满足 $q \geqslant 1$。前$\lfloor |r|/q \rfloor$个长度为 q 的子串称为 Q-sample，若最后一个子串的长度不足 q 则不是 Q-sample。

定理 1 对于给定的字符串 r，根据定义 3 进行分割，则 r 中任何一个长度为 l 的窗口内，至少含有 $\tau+1$ 个完整的 Q-sample。

定理 1 是显而易见的。通过定理 1 可知，字符串 r 中任何长度不小于 l 的子串 r^p 内至少包含 $\tau+1$ 个完整的 Q-sample。

定理 2 给定一个已按定义 3 分割的字符串 r，设 r^p 是 r 中的一个子串($|r^p| \geqslant l$)。给定另一个字符串 s，设 s^q 是 s 中的一个子串($|s^p| \geqslant l$)。若 $ed(r^p, s^q) \leqslant \tau$，则 s^q 中长度为 q 的子串至少能匹配到一个 r^p 中的 Q-sample。

证明：由定理 1 可知，r^p 中至少存在 $\tau+1$ 个完整的 Q-sample。因为 $ed(r^p, s^q) \leqslant \tau$，所以 r^p 和 s^q 的编辑距离不大于 τ。根据编辑距离原理，可通过不大于 τ 次编辑操作(插入、修改和删除)把 r^p 转换成 s^q。因为 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 2 成立。

定理 3(Q-sample 过滤器) 给定两个字符串 r 和 s，并采用定义 3 分割串 r。若$\langle r, s \rangle$为局部相似对，则串 s 中至少包含一个串 r 的 Q-sample。

定理 3 是定理 2 的推论和进一步总结。通过定理 3 可知，所有不满足定理 3 的串对一定不存在局部相似子串。因此，使用定理 3 作为过滤条件可以快速抛弃大量无关对。

3.2 局部相似验证和定位算法优化

LS-Join 算法[9]在验证过程中需要验证每一对匹配的 q-gram。验证过程中采用了一种双向扩展的验证方法，该方法对每个匹配的 q-gram 对分别计算其前向编辑距离矩阵和后向编辑距离矩阵。但如此验证也增加了验证的成本。本文将对 LS-Join 算法的双向扩展验证方法做进一步优化。

定义 4 给定一个按定义 3 分割的字符串 r，设 p_r 为字符串 r 中从第 p_r 个字符开始的 Q-sample 的位置(从 0 开始)。给定另一个字符串 s，设 p_s 为字符串 s 中从第 p_s 个字符开始的长度为 q 的字符串，称为 q-gram。若 p_r 位置的 Q-sample 与 p_s 位置的 q-gram 匹配，则称$\langle p_r, p_s \rangle$为一个 Q-sample 匹配。字符串 r 和 s 中所有的 Q-sample 匹配构成的集合称为 Q-sample 匹配集合，记为 $L_{\langle r,s \rangle}$。

q-gram 也是一个长度为 q 的子串，其不同于 Q-sample 的主要原因是：q-gram 是可重叠的(相邻 q-gram 重叠 $q-1$ 个字符)，而 Q-sample 是连续但不重叠的。

定理 4(无效 Q-sample 匹配) 给定两个字符串 r, s 和它们的 Q-sample 匹配集合 $L_{\langle r,s \rangle}$。若集合 $L_{\langle r,s \rangle}$ 中的某个 Q-sample 匹配$\langle p_r, p_s \rangle$满足 $p_r > p_s \wedge p_r - p_s > |r| - l + \tau$ 或 $p_r < p_s \wedge p_s - p_r > |s| - l + \tau$，则该匹配$\langle p_r, p_s \rangle$一定是无效的匹配。

证明：$p_r - p_s$ 是 Q-sample 匹配在编辑距离矩阵中的对角线编号。如图 1 所示，无效 Q-sample 匹配所在的对角线区域一定在两侧。该位置匹配的 Q-sample 由于向前或向后扩展少于 τ 个字符无法形成长度为 l 的子串，因此一定是无效的匹配。图 1 中，白色区域和浅灰色区域为可以扩展区域，而深灰色区域则是无法扩展的区域。

s \ r	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	2	3	4	5	6	7	8	9	10	11
1	-1	0	1	2	3	4	5	6	7	8	9	10
2	-2	-1	0	1	2	3	4	5	6	7	8	9
3	-3	-2	-1	0	1	2	3	4	5	6	7	8
4	-4	-3	-2	-1	0	1	2	3	4	5	6	7
5	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
6	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5
7	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4
8	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3

图 1 无效 Q-sample 匹配的位置示意图(深灰色区域)

Fig. 1 Positions of valid Q-samples (dark grey region)

定义 5 对于 Q-sample 匹配集合 $L_{\langle r,s \rangle}$，若集合 $L_{\langle r,s \rangle}$ 中的两个 Q-sample 匹配$\langle p_r^i, p_s^i \rangle$和$\langle p_r^j, p_s^j \rangle$满足 $p_r^i + q = p_r^j \wedge p_s^i + q = p_s^j$，则其称为连续匹配。$m$ 个连续匹配 Q-sample 可合并为一个 m 连续匹配。若把集合 $L_{\langle r,s \rangle}$ 中的所有连续匹配都进行合并，则将得到一个新的匹配区域集合 $M_{\langle r,s \rangle}$。设$\langle [b_r^i, e_r^i], [b_s^i, e_s^i] \rangle_{m_i}^{d_i}$为 $M_{\langle r,s \rangle}$ 中的一个匹配区域，其中 b_r^i 为第 i 个匹配区域在 r 中的起点，e_r^i 为第 i 个匹配区域在 r 中的终点，b_s^i 和 e_s^i 分别表示 s 中的起点和终点，m_i 为该匹配区域连续匹配的 Q-sample 数，d_i 为该匹配区域已含有的编辑距离，由 m 连续匹配构成的区域的 d_i 值为 0。

定理 5(连续匹配区域过滤定理) 给定两个字符串 r, s 和它们的匹配区域集合 $M_{\langle r,s \rangle}$。若匹配区域中最大的连续匹配数为 $m_{\max} = \max\{m_i\}$，且集合 $M_{\langle r,s \rangle}$ 中某个匹配区域$\langle [b_r^j, e_r^j], [b_s^j, e_s^j] \rangle_{m_j}^{d_j}$满足 $m_j < m_{\max} - (1 - 1/q)(\tau+2)$，则该匹配区域一定不是最长相似子串所在的区域。

证明：首先对含有 m_j 连续匹配区域的最大扩展范围进行理论分析，τ 编辑操作最多可向前或向后扩展 τ 个 Q-sample，同时双边 Q-sample 外还可能存在 $q-1$ 个字符相同，因此一个含有 m_j 连续匹配的区域最多可扩展到的区域长度为 $m_j q + \tau q + 2(q-1)$，如图 2 所示。而一个 $m_{\max}$ 连续匹配最短的扩展长度为 $m_{\max} q + \tau$。由 $m_j < m_{\max} - (1 - 1/q)(\tau+2)$ 可知，$m_j q + \tau q + 2(q-1) < m_{\max} q + \tau$，即定理 5 成立。

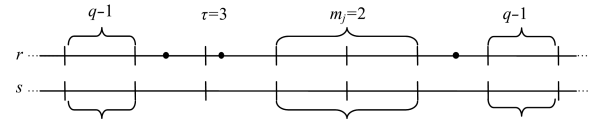


图 2 连续匹配最长可扩展匹配到的区域示意图

Fig. 2 Extend region of consecutive matches

定义 6 对于匹配区域集合 $M_{\langle r,s \rangle}$，若集合 $M_{\langle r,s \rangle}$ 中的两个匹配区域$\langle [b_r^i, e_r^i], [b_s^i, e_s^i] \rangle_{m_i}^{d_i}$和$\langle [b_r^j, e_r^j], [b_s^j, e_s^j] \rangle_{m_j}^{d_j}$满足 $ed([e_r^i + 1, b_r^j - 1], [e_s^i + 1, b_s^j - 1]) \leqslant \tau - d_i - d_j$，则称这两个区域为可合并区域。两个可合并区域合并后得到的新区域的

编辑距离为 $d_i+d_j+ed([e_r^i+1,b_r^j-1],[e_s^i+1,b_s^j-1])$。把 $M_{\langle r,s\rangle}$ 中的所有可合并区域都合并后得到一个新的区域集合 $G_{\langle r,s\rangle}$。集合 $G_{\langle r,s\rangle}$ 中具有 $v=\max\{e_r^i-b_r^i+1+\tau-d_i\}$（若 v 值相等取 $\min\{d_i\}$）的区域 $\langle[b_r^i,e_r^i],[b_s^i,e_s^i]\rangle_{m_i}^{d_i}$，记该区域 r 中最少可扩展的长度为 $l_{\min}=e_r^i-b_r^i+1+\tau-d_i$。

由于匹配区域集合 $M_{\langle r,s\rangle}$ 中的两个区域为间断区域，中间区域部分的编辑距离一定大于 0，因此当 $\tau-d_i-d_j\leqslant 0$ 时这两个区域一定不是可合并区域；又由于 r 中间隔的 Q-sample 数不能超过 $\tau-d_i-d_j$ 个，即当 $b_r^j-e_r^i-1>(\tau-d_i-d_j)q$ 时，这两个区域也一定不是可合并区域；此外，r 中间隔的长度与 s 中间隔的长度之差不能超过 $\tau-d_i-d_j$，即当 $(b_r^j-e_r^i-1)-(b_s^j-e_s^i-1)>\tau-d_i-d_j$ 时它们也一定不是可合并区域。

定理 6（非连续可合并区域过滤定理）　给定两个字符串 r，s 和它们的匹配区域集合 $G_{\langle r,s\rangle}$，$G_{\langle r,s\rangle}$ 中最少可扩展长度为 $l_{\min}$。对于 $G_{\langle r,s\rangle}$ 中的任意区域 $\langle[b_r^j,e_r^j],[b_s^j,e_s^j]\rangle_{m_j}^{d_j}$，若其最大可扩展长度满足 $(e_r^j-b_r^j+1)+(\tau-d_i)q+2(q-1)<l_{\min}$，则该区域一定不是最长相似子串所在的区域。

定理 6 的原理同定理 5，这里不再证明。根据定义 6，可以把满足要求的两个区域合并起来，同时用定理 6 去除不可能扩展成最长相似子串的区域。

4　基于 Q-sample 的局部相似连接算法

给定两个字符串集合 R 和 S、一个编辑距离参数 τ 和一个窗口长度 l。局部相似连接的定位问题是找出集合 R 和 S 中所有满足局部相似要求的字符串对，并给出最长相似子串的位置。本文提出了一种基于 MapReduce 框架和 Q-sample 的局部相似连接算法 MQLS-Join。为了实现局部连接的定位问题，MQLS-Join 算法设计了 3 个 MapReduce 阶段，即过滤阶段、验证阶段 1 和验证阶段 2。MQLS-Join 算法的框架如图 3 所示。

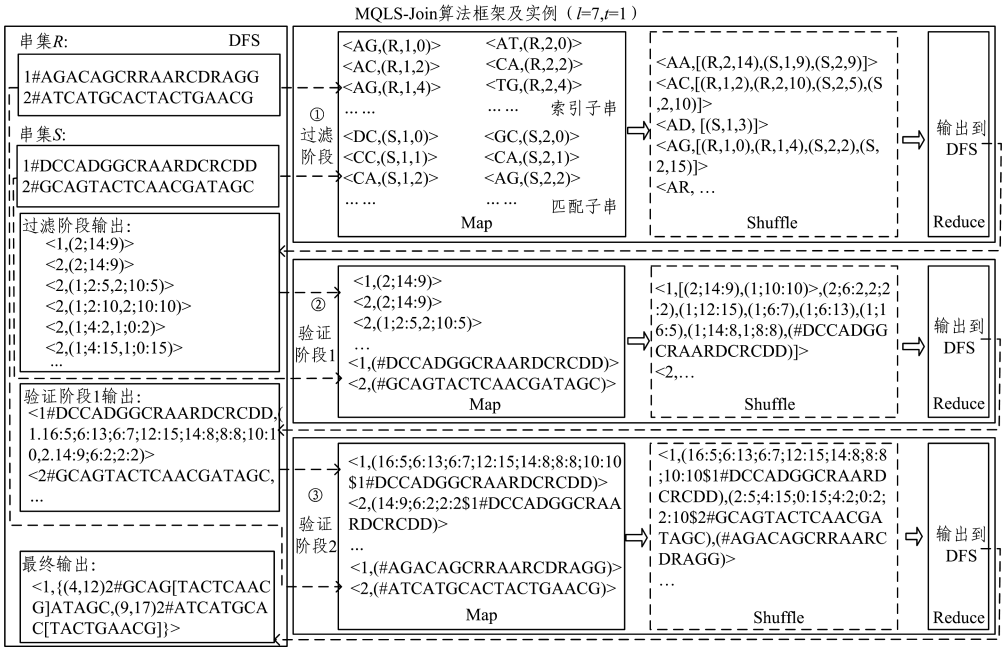


图 3　MQLS-Join 算法框架及实例

Fig. 3　Framework of MQLS-Join

4.1　过滤阶段

过滤阶段的主要目的是：避免枚举所有串对，而用条件快速过滤掉一定不含局部相似的无关对，并得到候选串对集，好的过滤条件能在较短的时间内抛弃大量无关对。为了实现快速过滤，MQLS-Join 算法采用了基于 Q-sample 的分割过滤策略。过滤阶段的输入为字符串集 R 和 S，它包含 3 个过程，即 map，shuffle 和 reduce。

1）map 过程：在 MapReduce 程序中并行且交替运行着众多 map 任务，每次 map 任务的输入是一个 key-value 对 $\langle sn,split\rangle$，其中 sn 是分片的编号，而 $split$ 是输入集合内容的一行内容。MQLS-Join 算法由于采用了基于 Q-sample 的分割方案，因此对来源于集合 R 或集合 S 的 $split$ 采用不同的分割方式。

分割集 R：若 $split$ 来源于集合 R，则将产生索引子串。首先根据“#”从 $split$ 中提取出字符串编号 rid 和串内容 r。然后采用定义 3 分割串 r 并获得 $\lfloor|r|/q\rfloor$ 个 Q-sample，其中 $q=\lfloor(l+1)/(\tau+2)\rfloor$。设 $r[p_r^i,p_r^i+q-1]$ 表示第 i 个 Q-sample，p_r^i 为起点，且 $1\leqslant i\leqslant\lfloor|r|/q\rfloor$，$p_r^1=0$，$p_r^i=(i-1)q$。最后输出这些 Q-sample 对应的索引子串的 key-value 对，即 $\langle r[p_r^i,p_r^i+q-1],(\text{‘}R\text{’},rid,p_r^i)\rangle$，其中 R 标识该项为索引子串。例子集合的部分索引子串如图 3 中过滤阶段的 map 过程所示。

$$\text{map}:\langle sn,split\rangle\rightarrow\langle r[p_r^i,p_r^i+q-1],(\text{‘}R\text{’},rid,p_r^i)\rangle$$

分割集 S：若 $split$ 来源于集合 S，则将产生匹配子串。首先根据“#”从 $split$ 中提取出字符串编号 sid 和串内容 s。然后对字符串 s 进行连续且重叠的 q-gram 拆分（相邻 q-gram 重叠 $q-1$ 个字符）。设 $s[p_s^j,p_s^j+q-1]$ 表示第 i 个 q-gram，其中 p_s^j 为起点，且 $1\leqslant j\leqslant|s|-q+1$，$p_s^1=0$，$p_s^j=j-1$。最后输出这些 q-gram 对应的匹配子串的 key-value 对，即 $\langle s[p_s^j,p_s^j+q-1],(\text{‘}S\text{’},sid,p_s^j)\rangle$，其中 S 标识该项为匹配子串。例

子集合的部分匹配子串如图 3 中过滤阶段的 map 过程所示。

map:$\langle sn,split\rangle \rightarrow \langle s[p_s^i,p_s^i+q-1],(\text{'}S\text{'},sid,p_s^i)\rangle$

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(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,例子集合的部分混淆排序结果如图 3 中过滤阶段的 shuffle 过程所示。

3)4educe 过程:在 MapReduce 程序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 $\langle sig,list((\text{'}R\text{'},rid,p_r^i)/(\text{'}S\text{'},sid,p_s^i))\rangle$,其中 sig 是子串,后面是该子串对应的所有索引子串和匹配子串的位置信息列表。首先根据位置信息中是否含有 R 把列表拆分成两个列表,即用 $Rlist$ 存储索引子串的位置信息,用 $Slist$ 存储匹配子串的位置信息。此时,如果 $Rlist$ 和 $Slist$ 中存在空列表,则可以断定在这个子串中不存在候选串对,因为在索引子串(集 R)和匹配子串(集 S)间不存在 Q-sample 匹配(定理 3)。如果两个列表均非空,则须生成两个列表所有项间的候选对(定理 3)。这里先循环列表 $Slist$,设 $Slist[j]$ 为第 j 个元素。为了确定 $Slist[j]$ 的所有候选对,需要完整遍历一次 $Rlist$ 列表,并把与 $Slist[j]$ 相关的所有候选对都添加到一个新列表 $list$ 中。若 $\langle Rlist[i],Slist[j]\rangle$ 是一个候选对,其中 $Rlist[i]=(rid,p_r^i)$,$Slist[j]=(sid,p_s^i)$,则把 $rid;p_r^i:p_s^i$ 添加到 $list$ 中。循环处理完 $Rlist$ 后,将生成 $Slist[j]$ 的一个候选对集的 key-value 对,即 $\langle sid,list(rid;p_r^i:p_s^i)\rangle$。如此循环处理完 $Slist$ 后,一次 reduce 任务结束。例子集合的部分 reduce 结果如图 3 中过滤阶段的 reduce 过程所示。

reduce:$\langle sig,list((\text{'}R\text{'},rid,p_r^i)/(\text{'}S\text{'},sid,p_s^i))\rangle -->\langle sid,list(rid;p_r^i:p_s^i)\rangle$

4.2 验证阶段 1

过滤阶段完成时,候选对集已生成。而验证阶段的主要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。但目前候选对集中仅有字符串的编号而无字符串的内容,因此无法进行验证。文献[7]中的 MassJoin 算法采用了二阶段读取的方法,使得每个字符串集在整个验证过程中只被读取一次。为了实现候选对集字符串内容的快速读取,本文算法采用 MassJoin 的读取方法,因此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 的主要目的是读取候选对中属于集合 S 的字符串内容,它的输入包括集合 S 和过滤阶段的输出结果。验证阶段 1 也包含 3 个过程,例子集合各过程的部分输出如图 3 中验证阶段 1 所示。

1)map 过程:在每次 map 任务中,如果读取的是集 S 中的串,则直接输出 key-value 对 $\langle sid,\#s\rangle$,其中 # 标识该项为字符串内容;如果读取的是过滤阶段的输出结果,则直接原样输出,即 $\langle sid,list(rid;p_r^i:p_s^i)\rangle$。

map:$\langle sn,split\rangle -->\langle sid,\#s\rangle$

map:$\langle sid,list(rid;p_r^i:p_s^i)\rangle -->\langle sid,list(rid;p_r^i:p_s^i)\rangle$

2)shuffle 过程:对 map 的输出按 sid 进行混淆、排序,并将结果作为 reduce 的输入。

3)reduce 过程:每次 reduce 任务的输入是一个 key-value 对,即 $\langle sid,list(list(rid;p_r^i:p_s^i)/(\#s))\rangle$。该输入中既包含串 sid 对应集合 R 的所有候选对信息,还包含串 sid 的内容 s。本文算法先循环遍历列表 $list(list(rid;p_r^i:p_s^i)/(\#s))$ 的所有项。若访问项的第一个字符是 #,则该项是串 sid 的串内容,将其保存以作备用;否则访问项是 $list(rid;p_r^i:p_s^i)$,此时循环处理列表 $list(rid;p_r^i:p_s^i)$ 中的每个 $rid;p_r^i:p_s^i$,即先提取串编号 rid 和匹配位置 $p_r^i:p_s^i$,再把 $p_r^i:p_s^i$ 添加到一个哈希集 $HS[rid]$ 中。如此处理完 $list(list(rid;p_r^i:p_s^i)/(\#s))$ 中所有的项。接着,遍历哈希集合 HS 中的每个 rid,同时提取该 rid 的匹配位置列表,并以 $rid.list(p_r^i:p_s^i)$ 格式作为一项添加到一个列表 $Clist$ 中,其中"."作为分隔 rid 和 $list(p_r^i:p_s^i)$ 的标志。最后,构建一个 key-value 对 $\langle sid\#s,Clist(rid.list(p_r^i:p_s^i))\rangle$ 并输出。

reduce:$\langle sid,list(list(rid;p_r^i:p_s^i)/(\#s))\rangle -->\langle sid\#s,Clist(rid.list(p_r^i:p_s^i))\rangle$

4.3 验证阶段 2

验证阶段 2 的主要目的是读取候选对中属于集合 R 的串内容并进行最终的验证定位工作。验证阶段 2 的输入包括验证阶段 1 的输出和字符串集合 R。验证阶段 2 依然包含 3 个过程,例子集合各过程的部分输出如图 3 中验证阶段 2 所示。

1)Map 过程:每次 map 任务根据输入源不同而进行不同的处理。若输入是集合 R 的字符串,则先提取串的编号 rid 和串的内容 r,然后输出 $\langle rid,\#r\rangle$,其中 # 代表本项是一个串内容。Map 过程的输入是验证阶段 1 的输出结果,即 $\langle sid\#s,Clist(rid.list(p_r^i:p_s^i))\rangle$。本文算法依次处理 $Clist$ 中的每个项 $rid.list(p_r^i:p_s^i)$。例如 $rid.list(p_r^i:p_s^i)$,先根据"."位置提取出 rid 和 $list(p_r^i:p_s^i)$,然后输出一个 key-value 对,即 $\langle rid,list(p_r^i:p_s^i)\ \$\ sid\#s\rangle$,其中 $ 是一个分隔符。如此,$Clist$ 中的每个项都输出了一个 key-value 对。

map:$\langle sn,split\rangle -->\langle rid,\#r\rangle$

map:$\langle sid\#s,Clist(rid.list(p_r^i:p_s^i))\rangle -->\langle rid,list(p_r^i:p_s^i)\ \$\ sid\#s\rangle$

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

3)Reduce 过程:每次 reduce 任务的输入是一个 key-value 对,即 $\langle rid,list((list(p_r^i:p_s^i)\ \$\ sid\#s)/(\#r))\rangle$。该 key-value 对中既包含串 rid 对应集合 S 的所有候选对信息,还含有该串的内容 r。该算法先循环遍历该列表的所有项。如果访问项的第一个字符是 #,则该项是串 rid 的串内容 r,将其保存以作备用;否则访问项是 $list(p_r^i:p_s^i)\ \$\ sid\#s$,需将其保存到一个新列表 $Dlist$ 中。循环处理完所有项后,此时已具备验证定位的条件。

为验证串 r 对应的每个候选对,需要循环处理列表 $Dlist$ 中的每个项。例如 $list(p_r^i:p_s^i)\ \$\ sid\#s$,先根据"$"和"#"的位置提取出串 r,s 间的 Q-sample 匹配集 $L_{\langle r,s\rangle}=list(p_r^i:p_s^i)$、集合 S 中串编号 sid 以及串内容 s。为了降低验证候选对的时间消耗,先进行如下优化:为去除 Q-sample 匹配集 $L_{\langle r,s\rangle}$ 中的无效 Q-sample,使用定理 4 进行过滤;针对过滤

后的 $L_{\langle r,s\rangle}$，根据定义 5 合并连续匹配，得到一个匹配区域集 $M_{\langle r,s\rangle}$，然后使用定理 5 过滤掉无效的连续匹配区域；接着针对过滤后的匹配区域集 $M_{\langle r,s\rangle}$，根据定义 6 合并所有非连续的可合并区域，然后使用定理 6 过滤掉无效的非连续可合并区域；最后得到过滤后的区域集 $G_{\langle r,s\rangle}$。

为了验证并定位候选对 $\langle r,s\rangle$，本文算法采用了 LS-Join 算法中的双向扩展验证方法[9] 验证区域集 $G_{\langle r,s\rangle}$ 中的每个 $\langle[b_r^i,e_r^i],[b_s^i,e_s^i]\rangle_{m_i^t}^{d_i}$，此时双向扩展的编辑距离总数不超过 $\tau-d_i$。设扩展后的区域为 $\langle[kb_r^i,ke_r^i],[kb_s^i,ke_s^i]\rangle_{m_i^t}^{kd_i}$，先计算该区域的评分 $v_i=ke_r^i-kb_r^i+1-kd_i$。如果区域长度满足 $ke_r^i-kb_r^i+1\geqslant l, ke_s^i-kb_s^i+1\geqslant l$，则该区域存在局部相似子串并保存 $\langle[kb_r^i,ke_r^i],[kb_s^i,ke_s^i]\rangle_{v_i}^{kd_i}$ 到扩展区域集 $H_{\langle r,s\rangle}$ 中。直到区域集 $G_{\langle r,s\rangle}$ 处理完后，如果集合 $H_{\langle r,s\rangle}$ 为空，则该候选对不存在局部相似；如果集合 $H_{\langle r,s\rangle}$ 非空，则从集合 $H_{\langle r,s\rangle}$ 中找一个评分最高的扩展区域，即取 $\max\{v_i\},1\leqslant i\leqslant|H_{\langle r,s\rangle}|$（当 $\max\{v_i\}$ 相同时取 kd_i 值最小的扩展区域）。最后，输出一个局部相似结果，即 $\langle kd_i,\{(kb_s^i,ke_s^i)sid\#s,(kb_r^i,ke_r^i)rid\#r\}\rangle$。

直到处理完 *Dlist* 的所有项，reduce 过程结束。

reduce：$\langle rid,list((list(p_r^i:p_s^i)\ \$\ sid\#s)/(\#r))\rangle-->\langle kd_i,\{(kb_s^i,ke_s^i)sid\#s,(kb_r^i,ke_r^i)rid\#r\}\rangle$。

5　实验

5.1　实验环境

为了验证本文算法的性能表现，在 Hadoop 平台上实现了本文的 MQLS-Join 算法。实验集群中的总节点数为 5 个（1 个主节点，4 个从节点），每个节点的硬件配置为：CPU i5 4590，内存 16GHz，硬盘 1TB。集群软件配置：操作系统 Ubuntu 18.04 64 位，Java 1.8，Hadoop 平台版本 2.7.1，开发环境 eclipse 4.8.0。为了对比本文算法与现有算法的性能差异，我们用 Java 实现了文献[9]中的 LS-Join 算法。由于 LS-Join 算法为内存算法，这里只将其运行在集群中的一个节点上。

本文实验数据来源于两个数据集，详情如表 2 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和标题的字符串。GBEST 数据集为 NCBI GenBank 的表达序列标签（Expressed Sequence Tags）[2]，实验去除了 EST 中的描述信息，只保留了序列本身。为了进行双集合局部相似连接实验，将表 2 中的数据集分割成了两个字符串数目相等的集合，即 R 和 S。

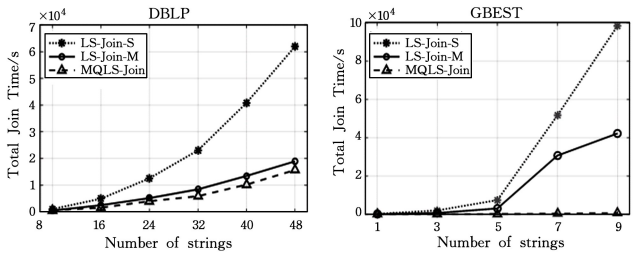
表 2　实验数据集信息

Table 2　Information of experiment datasets

数据集	Size/MB	Number of strings	Average length	Alphabet size
DBLP	45.6	480000	90.4	95
GBEST	31.2	90000	354.8	15

5.2　算法性能对比

在相似连接中，算法性能最重要的评价指标是相似连接的时间消耗。在进行两个集合局部相似连接时，除需要输入字符串集合 R 和 S 外，还需要给定窗口长度 l 和编辑距离 τ。文献[9]中 LS-Join 算法的最优配置参数 DBLP 集为 $l=50,\tau=5,q=7$，GBEST 集为 $l=100,\tau=7,q=9$。LS-Join 算法有两种实现方式：1）单线程的，记为 LS-Join-S；2）多线程的，记为 LS-Join-M。实验中由于实验机是 4 核心的，因此实验中线程数设为 4。为了在同样的数据集和参数下对比各算法的性能差异，本文的 MQLS-Join 算法在 DBLP 数据集上的连接参数为 $l=50,\tau=5$，在 GBEST 数据集上的连接参数为 $l=100,\tau=7$（本文算法不需要输入 q 值）。首先，对不同大小的字符串集分别采用 LS-Join-S，LS-Join-M 和 MQLS-Join 进行局部相似连接，并统计各种算法进行相似连接的总时间消耗。为了加快连接速度，LS-Join 算法为第一个数据集建立了倒排索引。因相似连接不同于近似匹配和信息检索，其主要目的是找出两个大数据集间的所有相似记录对，连接完毕时算法中建立的索引再无用途。因此，LS-Join 算法建立索引的时间也需记入总时间，即 LS-Join-S 和 LS-Join-M 算法的总时间包括建立索引时间和相似连接时间。MQLS-Join 算法的总时间包括过滤时间（验证阶段）和验证时间（验证阶段 1 和验证阶段 2）。随着数据集的增大，各算法的连接时间如图 4 所示。



（a）DBLP 数据集　　（b）GBEST 数据集

图 4　各数据集的大小与算法连接时间的关系

Fig. 4　Relationship of join time and size of different data sets

由图 4 可知，MQLS-Join 算法的相似连接速度在数据集较小时稍慢于 LS-Join-M 算法。这主要是因为 LS-Join-M 算法是一个多线程的内存算法，更适合小数据集的相似连接。但随着数据集的不断增大，MQLS-Join 算法的连接速度明显快于 LS-Join-S 和 LS-Join-M 算法，尤其在小字母表且长字符串的数据集（例如 GBEST）上算法性能更优。

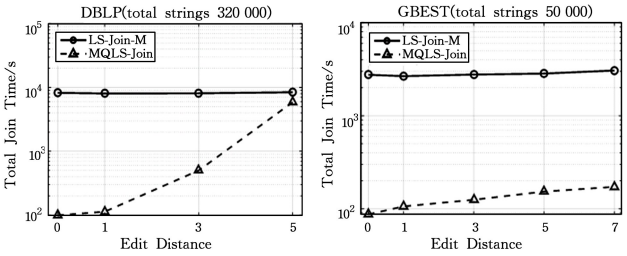
这主要是因为 MQLS-Join 算法是一个基于 MapReduce 框架的并行算法，它更适合大数据集的相似连接。另外，MQLS-Join 算法还通过去除无效匹配、合并连续匹配和合并非连续匹配区域等技术优化了双向扩展验证方法，这些优化在小字母表且长字符串的集合（例如 GBEST）上表现得更加高效。从图中还可以看出，随着数据集的增大，MQLS-Join 算法的连接时间基本呈线性增加。

本文还利用 LS-Join-M 算法和 MQLS-Join 算法在各数据集上分别从不同的编辑距离进行了相似连接实验。LS-Join-M 在 DBLP 数据集上的配置参数为 $l=50,\tau=0,1,3,5,q=7$，在 GBEST 数据集上的配置参数为 $l=100,\tau=0,1,3,5,$

1) http://dblp.uni-trier.de/xml/

2) ftp://ftp.ncbi.nlm.nih.gov/

7，$q=9$。MQLS-Join 在 DBLP 数据集上的配置参数为 $l=50$，$\tau=0,1,3,5$，在 GBEST 数据集上的配置参数为 $l=100$，$\tau=0,1,3,5,7$。随着编辑距离参数的增大，各算法的性能表现如图 5 所示。



(a)DBLP 数据集 (b)GBEST 数据集

图 5 各数据集上编辑距离大小与算法连接时间的关系

Fig. 5 Relationship of join time and edit distance of different datasets

由图 5 可知，LS-Join 算法由于 q 为定值因此受编辑距离参数的影响较小，且该算法中也没有适合所有编辑距离的 q 值。因此，针对不同数据集和编辑距离参数，需要实验测定最优 q 值，而且 LS-Join 算法在实际使用中不够灵活。而 MQLS-Join 算法随着编辑距离的增大，其连接时间也不断增加，这主要是因为 MQLS-Join 算法中计算的 q 值随编辑距离 τ 变化($q=\lfloor(l+1)/(\tau+2)\rfloor$)，即 τ 值越大，q 值越小，分割子串越多，处理时间越长。因此，MQLS-Join 算法在编辑距离参数变化的局部相似连接中比 LS-Join 更具优势。

结束语 本文研究的主要内容是基于两个字符串集的局部相似连接算法。本文提出了一种新的基于 MapReduce 框架的局部相似连接并行算法，解决了局部相似连接的定位问题。所提算法设计了一种基于 Q-sample 的字符串分割方案，获得了高质量的分割子串，减少了并行节点间的数据传输量。所提算法还使用了去除无效匹配、合并连续匹配和合并非连续匹配区域等技术，优化了 LS-Join 算法的双向扩展验证方法。实验结果显示，所提算法在大数据集上的相似连接速度快于当前的优秀算法 LS-Join。所提算法虽然通过并行技术加快了相似连接的速度，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。下一步将研究减少过滤阶段生成子串数量的方法，拟通过减少子串数量来进一步降低数据传输量；还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缩短验证时间。

参 考 文 献

[1] 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.

[2] 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.

[3] PAGH R. Large-scale similarity joins with guarantees[C]// Proceedings of 18th International Conference on Database Theory. Brussels, Belgium: Schloss Dagstuhl, 2015: 15-24.

[4] 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.

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

[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] 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. New York, USA: IEEE, 2014: 340-351.

[8] 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.

[9] WANG J Y, YANG X C, WANG B, et al. LS-Join: Local Similarity Join on String Collections[J]. IEEE Transactions Knowledge and Data Engineering, 2017, 29(9): 1928-1942.

[10] BAYARDO R J, MA Y, SRIKANT R. Scaling up all pairs similarity search[C]// Proceedings of 16th International Conference on World Wide Web. New York, USA: ACM, 2007: 131-140.

[11] 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.

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

[13] 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.

[14] WANG W, QIN J, XIAO C, et al. Vchunkjoin: An efficient algorithm for edit similarity joins[J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(8): 1916-1929.

[15] 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.

[16] 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, IN, USA: ACM, 2010: 495-506.

[17] 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.

[18] 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.

[19] 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.

[20] RONG C T, LIN C B, SILVA Y N, et al. Fast and Scalable Distributed Set Similarity Joins for Big Data Analytics[C]// Proceedings of IEEE 33rd International Conference on Data Engineering. New York, USA: IEEE, 2017: 1059-1070.