基于多核的细粒度并行的集合相似连接

荣垂田 李银银 冯林静 汪剑鸣

(天津工业大学 计算机科学与软件学院 天津 300387)

摘 要 相似连接是指在给定的两个数据集中,根据给定的相似性度量函数来计算数据之间的相似度,并找出所有相似度不小于给定阈值的数据对的操作.相似连接作为一个基本的操作,被广泛地应用于各种领域.随着网络和移动应用等信息技术的不断发展,数据呈现爆炸式增长,海量数据的分析需要强大的计算能力.为了满足不断增长的计算需求,提高计算效率和计算性能,计算机的体系结构也不断升级,出现了多核多处理器架构.为了提高相似连接的效率和计算资源的利用率,文中提出了基于多核的并行相似连接方法.相似连接操作与传统关系数据库中结构化数据之间的连接操作不同,相似连接处理的是异构数据,每一条数据处理的代价与其结构有关.为了实现多核处理器之间的任务均衡,文中提出了适合相似连接操作特点的数据长度均衡的数据划分方法.根据相似连接操作遵循 Filter-Refine 两阶段框架的特点,结合现代计算机体系结构的多核特性,提出了基于共享索引的任务分解方法和基于独立索引的任务分解方法.文中利用提出的数据划分方法和任务分解方法,实现了基于多核的并行化相似连接算法,包括自连接和 R-S 连接.文中对两种不同的实现方式的时间代价进行了分析,其中包括索引更新、索引扫描以及集合交运算的代价,从理论分析的角度证明了数据长度均衡划分和独立索引的实现方式在执行效率上具有优势.文中实验部分利用不同的数据集在多核多处理器平台上对并行相似连接的不同实现方式的执行效率和可扩展性进行了验证.实验结果证明,文中提出的数据长度均衡的数据划分方法以及基于独立索引的任务分解方法可以有效地提高并行相似连接的执行效率,在 16 核平台上使用 16 个线程在 DBLP 数据集上执行并行的相似自连接以及在 CiteSeer 和 DBLP 两个数据集上执行并行的 R-S 连接都可以在 1 秒内完成.

关键词 相似连接;并行;多核;多线程;数据划分

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.02320

Fine-Granular Parallel Set Similarity Join Based on Multicores

RONG Chui-Tian LI Yin-Yin FENG Lin-Jing WANG Jian-Ming

(*School of Computer Science and Software Engineering, Tianjin Polytechnic University, Tianjin* 300387)

Abstract Similarity join is a primitive operation that is widely used in many applications. It is used to find all similarity pairs whose similarity are not less than the given threshold from two datasets using the given similarity functions. As the development of information technologies, such as Internet and mobile applications et. al, the scale of the data set is increasing vastly. In order to analyze large scale data sets, it is needed to improve the computation capacity of computers. So, the architecture of computer with multi cores and multi processors has been invented to satisfy the increasing need of computation and to improve the computer's efficiency and performance. In order to improve the efficiency of similarity join and the utilization of computation resource, we proposed a solution for parallel similarity join based on multicores. In the relational database systems, the join operations are performed between structured data sets. While, the similarity join performs join operations between heterogeneous data sets. There is a big difference between

收稿日期:2016-05-09;在线出版日期:2017-01-04.本课题得到国家自然科学基金(61402329,61373104)和国家留学基金委资助.**荣垂田**,男,1981 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究领域为数据库与信息检索、云计算与大数据分析.E-mail:chuitian@tjpu.edu.cn.**李银银**,男,1992 年生,硕士研究生,主要研究方向为云计算与大数据分析.**冯林静**,女,1991 年生,硕士研究生,主要研究方向为数据库与信息检索.**汪剑鸣**,男,1974 年生,博士,教授,主要研究领域为计算机视觉、模式识别和大数据分析.

these two different operations. The cost of joining two records in relational database is the same in one query, as it performs joins on the same type of attributes. However, the cost of similarity join operation is related with the length of the attributes that to be joined. In order to improve the efficiency of similarity joins and utilize the computation capacity of multicores efficiently, we proposed to implement parallel similarity joins on multicores system using multi-threads technology. While, workload balancing is the assurance to get high performance for parallel programing. In order to achieve workload balancing among multicores, we proposed the even-length data partition strategy. As similarity join algorithms typically adopt a two-step filter-and-refine approach, we proposed two different task decomposition methods, shared-index based method and independent-index based method, regarding to the features of modern computer architecture. Based on our proposed data partition strategy and task decomposition methods, we implemented the parallel similarity joins based on multicores. We also analyzed the time cost of the two different implementations, even-quantity data partition with shared-index and even-length data partition with independent-index. In the time cost analysis, we considered the cost of index update, index scan and sets intersection. Based on our analysis, we proved that the implementation applying even-length data partition and independent-index can get better performance. In the experiment section, we conducted extensive experiments on four different implementations of parallel similarity joins using two different datasets. The experimental results showed that our proposed implementation applying even-length data partition strategy and independent-index can fully utilize the parallel computing features of multicores and get the highest performance. The efficiency of similarity joins can be improved vastly. The parallel self similarity join on DBLP data set and the parallel R-S similarity join between two data sets of CiteSeer and DBLP can be completed in a second using 16 threads on 16 cores system. We also compared our proposed methods with the state-of-art methods on the two datasets under the same experiment setups. The experimental results showed that our proposed methods can outperform the other methods by a wide margin.

Keywords　similarity join; parallel; multicore; multithread; data partition

1　引　言

在大数据时代，数据管理工作需要处理的是海量、多源、异构的数据，为了发掘大数据蕴藏的价值从而更好地为社会生产生活服务，需要对多源异构的数据进行数据集成和分析工作，其中一个重要的步骤是实现数据之间的关联，即从异构多源的数据集中找出描述同一个实体的所有数据. 由于数据来源的广泛性以及数据表达的多样性，数据的可用性非常差，表现为数据模式多样、数据不完整、数据重复、缺少唯一标示等. 由于大数据的以上特征，我们无法使用传统的关系数据库来完成数据之间的关联操作，这主要是由传统数据库本身的特征决定.

在传统的关系数据库系统中：(1) 所有的数据都必须满足提前定义的模式和完整性约束；(2) 数据之间的联系(关联)用表的主键和外键的值相同来实现；(3) 数据操作的结果都是基于逻辑判断关系的，即数据是否满足查询的要求只有两种可能：true(1)和 false(0). 因此，传统关系数据库系统擅长的是结构化数据上的精准语义的数据操作. 然而，由于应用环境的开放性以及数据精准语义的缺失，数据的可用性非常差，传统的关系数据库系统无法完成对多源异构数据的分析. 为了解决传统关系数据库的不足，人们开始研究高效的相似(近似)查询操作，包括近似查询和相似连接. 其中，相似连接是指根据用户指定的相似度计算方法和阈值从给定的数据集中找出所有相似记录对的操作过程.

根据相似连接操作处理的数据类型的不同，已有的相似连接的工作可以分为：字符串相似连接、集合相似连接、向量相似连接. 字符串相似连接将输入数据作为字符序列来处理，如人名、地名、邮编、DNA 序列等，字符之间存在严格的顺序，采用字符序列之间的编辑距离来衡量它们之间的相似度. 集

合相似连接将数据作为集合来处理,集合中的元素没有顺序的要求,如标题、文档、网页等,采用集合间公共元素所占的比例来衡量集合之间的相似程度.向量相似连接将输入数据作为向量来处理,如时空数据,图像、音频等多媒体提取的特征.向量的维度相同,向量的每一个分量代表意义不同,采用度量空间的方法来计算向量之间的距离以此来衡量向量之间的相似度.根据相似连接方法在执行的过程中是否会造成结果的丢失,已有的方法又可以分为精准的方法和近似的方法.无论哪种方法都有其特点和适用场景,要根据实际应用中处理的数据的特征和对处理结果的要求选择合适的方法.本文研究相似连接的目的是将其应用于文本重复检测和文本聚类,选择集合相似连接方法作为基本的操作来从数据集中找出所有相似的文本.尽管已经有很多集合相似连接方法相关的工作,但是它们关注的重点在于提出和发现复杂的算法来提高效率、没有考虑到现代计算机的体系结构已经发生了很大的变化,所设计的算法没有充分地利用计算机的资源和现代计算机体系结构的特点.基于以上的问题,本文研究的重点是在多核多处理器平台上实现高效的并行集合相似连接算法.

随着社会网络、移动应用以及物联网技术的发展和普及,数据产生的速度和规模得到了很大的提高.为了快速分析规模不断增长的数据,对计算机的性能提出了更高的要求.为了满足不断增长的计算需求,计算机核心处理器及体系结构已经做了很大的改进.随着处理器芯片集成度的不断提高,芯片的设计成本、能耗以及材料的物理特性都使得通过提高芯片集成度来提高处理器性能的做法已经行不通.为了进一步提高计算机的计算能力,出现了多核技术和多核多处理器架构(如图 1 所示),该架构通过将计算任务划分并分配到多个核上并行执行来提高执行效率.由 Pollack 定律可知,单个处理器的集成度提高一倍,最多能使其性能提高 40%;如果将两个处理器组合到一起构成一个双核处理器,使其硬件规模上与单个处理器提高一倍集成度相当的话,双核处理器可以实现 70%~80%的性能提升①;另外,多核多处理器架构可以有效地降低能耗.

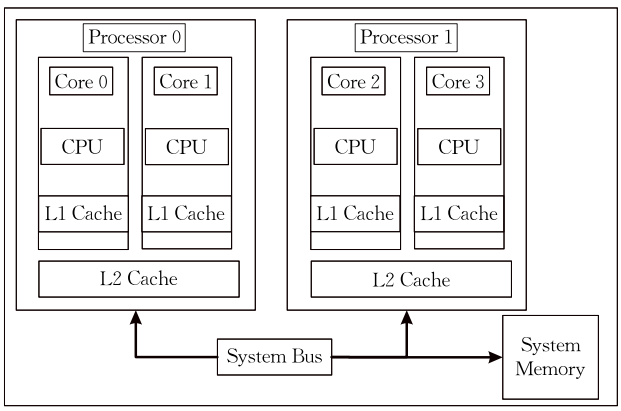


图 1 多核处理器架构

随着硬件技术的发展,计算机的硬件和体系结构已经发生了很大的变化,多核和多处理器技术已经在计算机中普及.但是,传统的算法实现没有充分应用硬件的新特性和体系结构的变化,一些观点认为操作系统和硬件技术本身已经很好地解决了多核之间的同步、缓存缺失等问题.实际上,大量的实验表明如果在算法的实现上充分考虑硬件平台的特征,可以显著地提高算法的效率[1-3].

根据我们的调查,已有的相似连接相关工作的研究重点在于提出复杂的算法和过滤机制[4-6],还没有相关的工作研究如何充分利用多核多处理器架构的并行处理特性实现并行的相似连接算法.因此,本文在深入理解集合相似连接操作特点的基础上,结合现代计算机体系结构的多核特性和并行化技术,做了以下几个方面的贡献:

(1)在数据划分方面,在分析结构化数据划分方法的基础上,根据非结构化数据的特征和相似连接操作的特点,提出了数据长度均衡划分方法;

(2)在任务分解方面,利用相似连接操作遵循的 Filter-Refine 两阶段框架的特点,结合多核多处理器的并行特性,对相似连接操作的执行过程进行分解,提出了共享索引的方法和独立索引的方法;

(3)根据提出的数据划分和任务分解方法,在多核多处理器平台上实现了基于共享存储的细粒度并行的相似连接算法;

(4)在实验方面,对相似连接方法并行化的不同实现方式进行了验证,包括相似自连接和 R-S 连接的性能、可扩展性.

本文按以下的顺序进行论述,在第 2 节论述已有的相关工作;第 3 节是问题定义和基础知识;第 4 节论述数据划分和任务分解方法;第 5 节论述基于多核的并行相似连接的实现细节及优化策略;第 6 节通过实验对本文提出的方法进行验证;最后是对

① http://www.ibm.com/developerworks/cn/aix/library/au-aix-multicore-multiprocessor

本文研究的总结和未来工作的设想.

2　相关工作

在许多领域，相似连接操作得到了广泛应用，如数据清洗[7]、实体识别[8-9]、数据集成[10-12]等. 由于应用环境的变化以及精准语义的缺失，现在越来越多的应用转向近似的语义来处理数据. 因此，相似连接的工作在近年来成为学术界和工业界关注的热点[13-15].

2.1　传统的方法

为了提高相似连接操作的执行效率，已有的相似连接方法都遵循一个 Filter 和 Refine 的两阶段框架. 在 Filter 阶段，通常都包含数据分组或划分的过程，并且会利用数据之间相似的必要条件来对数据进行过滤，以此来尽可能少地产生候选集. 在 Refine 阶段，候选集中所有可能的数据对会被验证是否为最终的结果. 由于不同的相似连接方法所使用的数据划分方法和过滤机制不同，有的方法可能会丢失真正的结果. 根据相似连接算法的结果是否有缺失的可能，已有的相关工作可分为两类：精准方法和近似方法. 同时，根据相似连接方法所针对数据类型的不同，已有的相关工作还可分为基于集合的方法 SSJoin[13-17]和基于字符串的方法 EDJoin[18-19]. SSJoin 将数据看成由若干元素(token)组成的集合，利用集合之间的共同元素所占的比例来衡量数据之间的相似度；EDJoin 将数据看成字符的有序序列，利用字符序列之间的编辑距离来衡量数据之间的相似度. 本文研究的重点是将精准的 SSJoin 在多核多处理器平台上进行并行化方法，下面论述精准的 SSJoin 和多核多处理器平台相关的工作.

SSJoin 方法基本上都使用了前缀过滤的思想. 为了更好地利用前缀过滤的思想，在执行相似连接过程之前需要有一个对数据集预处理的过程. 首先，它需要把数据集中的每一条数据切分成 token 的集合，这里的 token 可以是单词或者 N-Gram 的形式. 然后，对生成的集合使用统一的排序规则进行排序. 在排序后的 token 序列中，根据其中元素的多少和相似度阈值，提取若干个 token 作为集合的前缀. 根据集合间覆盖关系的必要条件可以知道，相似的集合之间至少有一个公共的 token 出现在它们各自的前缀中. 最后，将数据集中的数据根据其对应的集合中元素个数递增的次序进行排序，以保证相似的数据分布在临近的位置上，以更好地利用程序的局部性原理. AllPairs[14]在预处理后的数据集上，利用所有数据的前缀构建倒排索引，相似的数据肯定会出现在同一个倒排列表中. 倒排索引中每一个倒排列表的索引项的顺序与预处理后的数据一致，即相似数据的索引项在同一个倒排列表中也是临近的. 因此，在扫描倒排列表的过程中可以利用集合间相似的必要条件，即长度约束(集合的大小)来减少倒排列表的扫描范围. PPJoin[15]同 AllPairs 方法一样为所有数据的前缀构建了倒排索引，它的创新在于利用数据的公共前缀 token 的位置信息来估算它们的相似度上界，根据此上界是否不小于相似度阈值来选择可能相似的数据，与 AllPairs 相比可以过滤掉更多的不可能相似的数据，减少了最后验证的开销. PPJoin+[15]不仅利用了公共前缀 token 的位置信息，还利用了数据的后缀的位置信息. 此方法会从一条数据的后缀中选出位于中间位置的 token，并利用此 token 去划分另一条数据的后缀，然后根据 token 将两条数据划分成片段的大小来估算后缀部分公共元素的多少，与前缀部分判断的结果一起来估算相似度的上限. PPJoin+方法中使用的过滤方法可以大大减少候选集的大小，有效提高相似连接的效率. AdaptJoin[17]方法突破以往固定前缀长度的做法，提出了可变长前缀过滤的方法和构建增量索引的机制，通过对前缀过滤思想的灵活应用，进一步提高了相似连接的效率. 文献[20]根据相似数据之间的长度约束以及鸽巢原理，提出了一种基于集合划分的相似连接方法，此方法可以有效地提高过滤的效率. 文献[21]根据前缀过滤与全局排序规则无关的特点，提出了一种基于多种排序规则的相似连接方法. 文献[22]对多属性上的近似查询进行了研究，基于前缀的思想提出了前缀树的索引来优化相似连接的执行过程，并给出了代价模型来指导前缀树的构建.

2.2　基于新硬件特性的方法

为了不断提高计算机的性能，计算机的体系结构不断升级，出现了大内存、多核多处理器、GPU 等新的硬件设备，为基于内存的实时在线数据分析提供了基础.

文献[1]中对传统等值连接操作利用 Sort-Merge 方法在大内存和多核平台上进行了改进，提出了大规模并行的 MPSM 算法. 此算法通过避免细粒度的同步和远程的 NUMA 内存分区的访问来实现高性能和线性可扩展性. 文献[2-3]在多核平台上对 Sort-Merge 和 Radix-Hash 两种连接算法的实现方法进行了实验分析，实验结果与传统的 SIMD 和

NUMA 机制下不同.传统的实现方式在 SIMD 和 NUMA 机制下 Sort-Merge 优于 Radix-Hash,但是在多核平台上通过利用硬件新特性的实验表明 Radix-Hash 在绝大数情况下优于 Sort-Merge 方法.文献[23]将文献[24]中的基于划分机制的 ED-Join 算法和过滤方法扩展到多核平台上,实现了并行的 ED-Join,算法的执行效率得到了大幅提升.文献[25]将向量之间的连接问题转换为 GPU 的 Sort-and-Search 操作,通过利用多个空间填充曲线的方法将数据降维,以此减少数据的搜索空间.实验结果表明文献[25]中的 LSS 方法在 1 百万数据量的情况下,比单进程的 GESS 方法提高 5.5～118.3 倍,比单进程的 GEO 方法提高 3.6～26.9 倍.

通过相关的实验证明,程序的并行化结合多核多处理器的并行处理特点可以有效地提高程序的执行效率.但是,还没有文献研究集合相似连接方法如何充分利用多核多处理器平台并行特性来实现细粒度的并行算法.因此,本文研究基于多核的并行集合相似连接,通过程序的并行化来充分利用多核多处理器的并行化特性以此提高集合相似连接的效率和处理大规模数据的能力.

2.3 基于 MapReduce 的方法

随着数据规模的不断增加,数据管理和分析所面临的挑战不断增加,由于 MapReduce 具有可扩展性和高可用性的特点,成为大数据分析的有力工具.

文献[26]将文本数据看成 token 集合的形式,对基于 MapReduce 的集合相似连接进行了研究,实现了文献[15]中提出的 PPJoin+方法,其中包含 3 个 MapReduce 阶段.文献[26]根据前缀过滤的思想,将每一条记录的前缀中包含的 token 作为记录的 key,记录本身作为 value 输出,所有可能相似的记录会被汇集到同一个 Reduce 节点上进行验证.文献[26]的缺点在于每一条数据根据其长度以及相似度阈值的不同,会有若干个 token 被选为前缀,因此每一条记录会被输出多次,并且由于很多记录会有两个或者两个以上共同的前缀 token,因此在整个计算过程中会造成大量的数据冗余和重复计算.

文献[27]提出了一个包含 Join 和 Similarity 两个阶段的 V-SMART-Join 方法,此方法在两个阶段通过逐渐聚集的方式来完成相似连接.在 Join 阶段,将每一条记录包含的所有 token 作为 key 输出,然后进行汇聚得到类似倒排列表形式的输出.在 Similarity 阶段,通过两个 MapReduce 任务处理 Join 阶段产生的类似倒排列表的结果.通过枚举倒排列表中的每一个记录对,得到所有包含共同 token 的记录对;得到的记录对进一步聚集,最终得到所有记录之间的交集.文献[27]从 token 的级别通过多个 MapReduce 任务逐渐地聚集得到最后的结果,此方法的缺点在于枚举倒排列表中的记录对会产生大量的中间结果,数据的 Shuffle 过程开销非常大,对节点内存要求比较高;计算的过程中没有使用过滤方法将不必要的中间结果提前过滤掉而造成大量的不必要计算;另一缺点是需要多次从分布式文件系统输入和输出数据.

文献[28]提出的 Mass-Join 方法,将文献[24]中基于划分的相似连接方法进行了扩展,利用 MapReduce 进行实现.在 Mass-Join 方法中,首先根据鸽巢原理将数据集 R 中字符串进行均衡划分,并抽取其中的部分片段作为记录的 Key.对于数据集 S 中的字符串,为了保证与 R 中相似的字符串生成相同的 Key,对于数据集 S 中长度为 L 的字符串,它需要为长度在 $\lceil L*\theta \rceil$ 到 $\lfloor L/\theta \rfloor$ 之间的每一个整数(长度),抽取其中的若干子串作为其 Key.如果字符串长度为 100,$\theta=0.8$,可见仅 S 中的一个字符串产生的 Key 的数量.文中对中间结果生成的倒排列表进行的枚举处理效率相当低下.尽管文中提出了一些 Key 合并的策略,在整个计算过程中大量的数据重复是影响性能的关键因素.

文献[29]针对数据分布不均衡的情况,利用数据划分的方法实现了 MapReduce 计算节点之间的负载均衡.文中首先利用一个 MapReduce 任务对数据进行预处理得到 Key 和数据块分布的统计信息矩阵;然后在另一个 MapReduce 任务中的 Map 阶段利用 BlockSplit 或 PairRange 对信息矩阵进行划分以保证 Reduce 节点之间的负载均衡.文中的关键在于,为划分后的矩阵块及其对应的数据生成 Composite Key(组合关键字)利用 MapReduce 中的 Partitioner 和 Comparator 方法控制数据 Shuffle 的过程来实现 Reduce 节点上的负载均衡.

综上所述,在不同的计算平台上,相似连接方法的实现方式不同,研究的重点不同,适用的应用场景不同.传统的方法(或单进程的方法)专注于发明更加高效的过滤方法和索引机制,通过减少不必要的计算来提高算法的效率.基于新硬件的方法,致力于研究程序的并行化实现方式,通过充分利用硬件的并行化特性来提高程序的并行化程度和执行效率.以上两类方法适用于基于内存的实时在线分析.基于 MapReduce 的方法,研究相似连接操作在

Shared-Nothing 大规模集群上高效的实现方式，此类方法主要是应用于大规模数据的离线批量处理. 本文研究的重点是多核多处理平台上细粒度并行的集合相似连接方法.

3　问题定义和相关知识

相似连接是一种基本的操作，它利用给定的相似度计算方法从两个数据集中找出所有相似度不小于给定阈值的记录对. 本文将数据集中的记录看成是有若干元素(token)组成的集合，并利用基于集合的相似度计算方法计算记录之间的相似度. 为了论述的方便，本节给出问题的定义及所使用的术语和相关的背景知识. 本文使用的符号统一在表 1 中进行了描述.

表 1　文中使用的符号及定义

符号	定义
Σ	字符集
s	由 Σ 中的字符组成的字符串
S	字符串集合
B	数据块
$\lvert\cdot\rvert$	集合中元素个数
t	字符串 s 中的元素(token)
I_t	token t 对应的倒排列表
T_s	字符串 s 中包含的元素集合
T_s^p	字符串 s 的前缀 token 集合
U	数据集 S 包含的 token 集合 $U=\{\bigcup T_s(s\in S)\}$
θ	相似度阈值
$sim(s_i,s_j)$	字符串 s_i 和 s_j 之间的相似度
O	U 中 token 的一个全序关系

3.1　问题定义

定义 1.　Token 集合.

假设 s 是一条数据(字符串，字符来自 Σ)，s 的 token 集合定义为 $T_s=\{t_1,t_2,\cdots,t_m\}$，其中 t_i 是将 s 利用划分方法划分后得到的一个片段.

定义 2.　Token 域.

对数据集 S 中每一条数据 s，利用统一的划分方法将每一条数据划分成 token 集合. 数据集 S 所包含的所有 token 组成的集合称为数据集 S 的 token 域 U.

$$U=\bigcup_{i=1}^{|S|}\{t\mid t\in T_s\},\ s\in S.$$

定义 3.　全局序列.

对于数据集 S 的 token 域 U，U 上的一个全序关系，记为 O. 例如，token 在 S 中出现的频率大小关系、字典排序的顺序等.

定义 4.　规范化.

将数据集 S 中的每一条数据 s 划分后得到的 token 集合 T_s 按照 O 定义的全序关系进行排序，此过程称为规范化 $C(s,O)$.

定义 5.　相似度计算方法.

相似度计算方法用来计算数据之间相似的程度. 本文适合的 3 种基于集合的相似度计算方法 Jaccard，Dice 和 Cosine.

$$\mathrm{Jaccard}(r,s)=\frac{|T_s\cap T_r|}{|T_s\cup T_r|},$$

$$\mathrm{Dice}(r,s)=\frac{2|T_s\cap T_r|}{|T_s|+|T_r|},$$

$$\mathrm{Cosine}(r,s)=\frac{|T_s\cap T_r|}{\sqrt{|T_s|*|T_r|}}.$$

定义 6.　长度约束.

如果两条数据 r 和 s 相似，即 $sim(r,s)\geq\theta$，r 和 s 对应的 token 集合一定满足长度约束. 定义 6 中的相似度计算方法对应的长度约束如表 2 所示.

表 2　相似度计算方法的性质

相似度公式	长度约束	前缀大小
Jaccard	$\lceil\lvert T_s\rvert\times\theta\rceil\leq\lvert T_r\rvert\leq\lfloor\lvert T_s\rvert/\theta\rfloor$	$\lvert T_s\rvert-\lceil\lvert T_s\rvert*\theta\rceil+1$
Dice	$\left\lceil\lvert T_s\rvert\times\frac{\theta}{2-\theta}\right\rceil\leq\lvert T_r\rvert\leq\left\lfloor\lvert T_s\rvert\times\frac{2-\theta}{\theta}\right\rfloor$	$\lvert T_s\rvert-\lceil\lvert T_s\rvert*\theta\rceil+1$
Cosine	$\lceil\lvert T_s\rvert*\theta^2\rceil\leq\lvert T_r\rvert\leq\lfloor\lvert T_s\rvert/\theta^2\rfloor$	$\lvert T_s\rvert-\lceil\lvert T_s\rvert*\theta^2\rceil+1$

定义 7.　前缀.

$\forall s\in S$，$L_s=C(s,O)$，O 为全局排序规则，θ 为相似度阈值. 以 Jaccard 为例，记录的前缀定义为

$$T_s^p=\{t_i\mid t_i\in T_s,1\leq i\leq|T_s|-\lceil|T_s|*\theta\rceil+1\}.$$

定义 6 中的相似度计算方法所对应的前缀集合的大小如表 1 所示.

定义 8.　相似连接.

对于数据集 R 和数据集 S，相似连接操作根据用户指定的相似度计算方法 sim 和阈值 θ，从数据集 R 和数据集 S 中找出所有满足 $sim(r_i,s_j)\geq\theta$ 的记录对$\langle r_i,s_j\rangle$.

3.2　基于前缀的索引

SSJoin 方法的出发点在于相似的数据应该包含共同的元素(token). 如果利用数据集中的 token 构建倒排索引，那么相似的数据必定会出现在同一个倒排列表中. 通过扫描倒排列表就能找出所有相似的数据. 但是，利用数据集中的所有 token 构建的

倒排索引规模太大，并且由于相似的数据之间往往包含大部分相同的 token，它们会共同出现在多个倒排列表中，那么扫描索引的代价还是比较高. 为了减小索引的规模，提高索引扫描的效率，在构建倒排索引时可通过利用前缀的性质来实现. 两条数据 r 和 s，经过同一个全序 O 规范化后的，如果 $sim(r,s)\geq\theta$，那么 $T_r^p \bigcap T_s^p \neq \varnothing$，这是两个数据相似的必要条件，也就是说 r 和 s 的前缀中一定包含至少一个共同的 token. 那么根据此性质，在构建倒排索引时只需要利用每一条数据的前缀部分即可保证相似的数据出现同一个倒排列表中. 用来对数据进行规范化的全序 O 一般都采用数据集中词频升序的方式. 利用词频升序的排序方法可以保证每一条数据的前缀都是本条数据中出现次数较小的若干个，利用此方法产生的前缀构建的倒排索引规模最小. 由于数据集经过预处理之后是按照数据长度递增的次序排序的，根据相似记录之间的长度约束可知相似的记录排在临近的位置. 那么，基于预处理后的数据集构建的索引依然保证相似的记录在同一个倒排列表中也是临近的，可以更好地利用程序的局部性原理.

3.3 多核多处理器架构与并行计算

随着计算机技术在生产生活中应用的普及，计算机所处理数据的规模在不断地增加，人们对计算机性能的要求在不断地提高，处理器的结构在不断地升级以满足人们对计算能力的需求.

以 Intel 为例，处理器从最初的 8 位处理器 Intel 8088，通过不断发展，1978 年出现了 16 位处理器 Intel 8086，1985 年 Intel 公司推出了 80X86 系列的第一款 32 位处理器，2001 年推出了 64 位处理器. 处理器的主频从几十 MHz 发展到今天的 4GHz. 处理器的设计成本、功耗以及散热问题成为制约计算机性能提高的瓶颈. 为了解决以上的问题，Intel 于 2005 年推出了双核处理器，此后出现了多核多处理器架构. 此架构在性能、能耗以及可扩展性方面具有很好的优势，因此现代的计算机系统普遍采用此架构. 如图 1 所示，一个具有两个处理器、每个处理器有两个核心的多核多处理器系统，每个核心可以支持两个硬件线程. 每个核心有一个 L1 缓存，同一个处理器上的核心可共享 L2 缓存. 在最新的处理器系统中，每个核心可能都拥有自己的 L2 缓存.

在多核多处理器平台上对应用程序并行化的实现方式有两种，多进程和多线程. 由于进程的调度和进程之间的通信代价比较高，使得多进程不适合用于开发细粒度并行的应用程序. 自操作系统引入多线程技术以后，线程是任务调度的基本单位，进程是资源的拥有单位. 线程创建、调度、切换的代价远小于进程. 再加上在多核多处理器系统可以同时支持多个硬件线程[30-31]. 因此，多线程技术适用于实现基于共享存储的细粒度并行应用程序. 在多核多处理器平台上对应用程序进行并行化，主要任务是根据它们在多核多处理器平台上的执行方式实现对数据的划分和任务的分解，使得分解后的各分任务之间尽可能地均衡和独立，然后利用并行编程技术将任务并行化实现.

为了衡量程序并行化的效果，一般采用加速比来衡量. 加速比(speedup)是指完成相同的任务在单处理器系统运行所消耗的时间与在并行处理器系统中运行所消耗的时间之间的比率.

4 数据划分和任务分解

在多核多处理器平台上实现并行任务，关键是要对数据进行划分和对任务进行正确的分解，使得各个处理器及其处理核心上的负载尽可能均衡，分解后的任务尽可能独立，以此来尽可能利用计算资源提高任务的执行效率. 本节根据相似连接操作的特点，论述其在多核多处理器平台上实现并行化所采取的任务分解和数据划分方法. 在本论文中实现的是共享内存的并行化算法，任务所要处理的数据都在内存中，线程的数量与数据的划分数量相同.

4.1 数据划分方法

由于相似的数据之间的长度满足长度约束条件，为了利用程序的局部性原理以及前缀的性质，首先会对数据集进行规范化，然后按照数据长度递增的次序进行排序，相似的数据会分布在临近的位置上. 本文使用的全局排序规则 O 为 token 在集合 S 中出现的次数的升序. 数据的划分是在排序后的数据集上进行的.

(1) 数据量均衡划分

在传统的数据库和数据仓库中，由于数据都是结构化的，最常用的数据划分方法是数据量均衡划分，即把数据集划分成包含数据量相同的若干块. 针对结构化数据的分析任务，对每一条数据的操作都是相同的，因此数据量均衡划分能保证数据分析任务在每一个数据块上消耗的代价相同. 数据量的均衡划分如下面的式(1)所描述，其中 $B_1, B_2, \cdots, B_N$ 是对数据集 S 进行等量划分后得到的 N 个数据块，

每一个数据块中的数据量相同.

$$\begin{cases} B_1 \cup B_2 \cup \cdots \cup B_N = S \\ |B_i| \approx |B_j| \approx \dfrac{|S|}{N} \\ B_i \cap B_j = \varnothing \end{cases} \tag{1}$$

传统的关系数据库处理和分析的都是结构化数据，每一条数据的结构相同，每一条数据作同样操作的代价也相同. 因此，关系数据库在并行化处理的过程中所采用的数据量均衡划分方式实现了等工作量划分.

与传统的结构化数据分析的任务不同，相似连接操作所处理的数据是异构的数据，每一条数据进行相似连接操作的代价不同. 实际上，处理每一条数据的代价与数据本身的长度和相似度阈值有关. 在相似连接处理的过程中，每一条数据的前缀会被选取用来扫描相应的倒排列表去获得可能与之相似的数据. 在相同相似度阈值的情况下，数据的前缀中包含 token 的多少与数据的长度有关，数据的长度越长其前缀中的 token 越多(定义 5)，也就意味着需要扫描更多的倒排列表，获得更多候选数据去进一步验证. 因此，在相似连接并行化的过程中采用数据量均衡划分方法会造成不同的处理器上的负载相差较大.

(2) 数据长度均衡划分

$$\begin{cases} B_1 \cup B_2 \cup \cdots \cup B_N = S \\ \sum_{m}^{|B_i|} |s_m| \approx \sum_{n}^{|B_j|} |s_n| \approx \dfrac{\sum_{k}^{|s|} |s_k|}{N} \\ s_m \in B_i,\ s_n \in B_j,\ B_i \subset S,\ B_j \subset S,\ s_k \in S \end{cases} \tag{2}$$

为了尽可能地实现不同处理器之间的负载均衡，对于相似连接操作这样的任务需要根据其数据处理的特点进行数据划分. 由于相似连接过程中，每一条数据进行处理的代价与数据本身的前缀中包含 token 的多少成正比，即同一相似度阈值的情况下与数据本身的长度成正比. 因此，可以按照数据块中数据的长度之和尽可能相近的规则来对数据集进行划分，以此来保证不同处理器上的负载均衡. 如式(2)所述，数据集 S 中数据的总长度为 $\sum_{k=0}^{|S|} |s_k|$，每一个数据块 B_i 包含长度之和近似为 $\sum_{k=0}^{|S|} |s_k| \Big/ N$ 的数据，N 为数据集 S 需要划分的数据块的个数. 式(2)中的近似是为了保证每一个数据块中的数据的完整性.

4.2　任务分解方法

由进程和线程之间的关系可知，由同一进程派生出的多个线程会共享进程的空间和资源，线程本身还有独立的子空间. 由于多核多处理器平台上的每个处理器核心都可以支持硬件线程，线程之间可以实现真正的并行执行. 本文提出了两种不同的任务分解方法：(1) 共享索引的方法，其中进程利用全部数据创建索引，线程间共享该索引(如图 2 所示)；(2) 独立索引的方法，进程只负责数据的划分，线程利用分配的数据块及相关数据创建独立的索引(如图 3 所示).

(1) 共享索引的方法

共享索引的方法中，进程利用整个数据集构建全局的索引，由此进程派生出的线程会共享此索引. 然后，进程根据用户提供的信息为每个线程分配数据块和需要扫描的共享索引的范围，进程派生线程的数量与数据划分个数相同. 派生的多个线程根据其分配得到的数据块及共享索引的扫描范围，在多个处理器核心上并行执行相似连接任务. 如图 2 所示，数据、索引及多个线程属于同一个进程空间，进程负责构建索引及数据划分任务，多个线程可以访问进程空间内的数据块和索引，线程有独立的子空间会被系统调度到不同的处理器核心上并行执行.

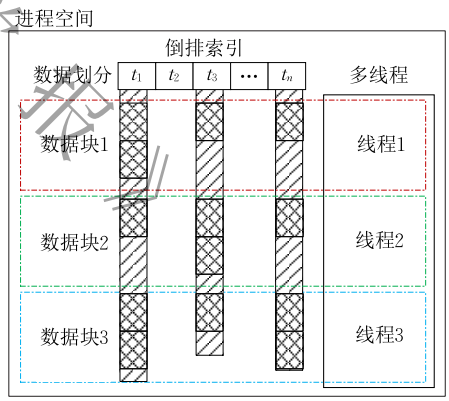


图 2　共享索引的方法

(2) 独立索引的方法

独立索引的方法中，进程根据用户指定的并行线程的数量对数据进行划分，然后派生出与数据划分数量相同的线程. 线程利用分配的数据块及相关的数据在自己的子空间构建独立的索引，如图 3 所示. 线程利用分配的数据块构建的索引在线程的子空间内，不能为其它线程所直接访问. 数据和多个线程在同一个进程空间，独立的索引和线程任务在同一个线程子空间. 这样的任务分解可以进一步提高线程之间的独立性.

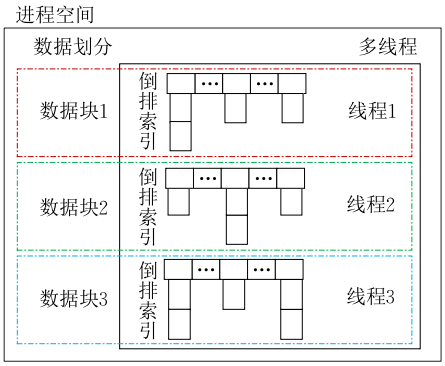


图 3 独立索引的方法

5 基于多核的并行相似连接

本文以 PPJoin+[15] 方法核心为基础，如算法 1 所描述. PPJoin+算法主要包含 3 个步骤，构建索引、扫描索引以及验证. 在扫描索引的过程中，为了减少候选集合的大小，采用了 3 种过滤方法，长度过滤、前缀位置过滤和后缀位置过滤. 根据相似数据的长度约束可知，数据集经过预处理后将相似的数据排在临近的位置. 如果采取正确的数据划分方法，将相似的数据划分到同一个数据块中，以上的 3 个步骤都可以并行地执行. 在深入分析相似连接执行特点的基础上，通过利用任务分解和数据划分方法，结合多核多处理器平台的并行处理特性，本文实现了基于多核的并行相似连接方法. 根据任务分解方法的不同，本节分别论述并行相似连接的具体实现.

5.1 共享索引的并行相似连接

基于共享索引的并行相似连接方法，进程负责构建全局索引和数据划分，多线程根据分配的数据块和共享索引的范围并行地执行相似连接任务. 由于数据在划分之前进行了规范化处理及排序，相似的数据分布在临近的位置. 划分之后，不同的数据块由不同的线程在不同的处理器核心上进行处理. 由于相似的数据可能会分布在相邻的块中，如果线程仅处理进程分配的数据块及由本数据块构建的索引范围的话会导致一部分结果丢失. 为了解决这个问题，本文利用相似数据之间的长度约束为每个线程分配共享索引的扫描范围，处理相邻数据块的线程之间所要扫描的索引部分会有部分重合 Δ_1 和 Δ_2，如图 4 所示. Δ_1 和 Δ_2 的计算方法如下面的两个公式所描述，其中 I_t 为元素 t 所对应的倒排列表，I_t^i 为由来自数据块 B_i 所包含数据的前缀所构建的倒排列表 I_t 中相对应的部分，$I_t[j]$ 为 I_t 中的一个记录项，$S_{I_t[j]}$ 为倒排列表 $I_t[j]$ 所对应的数据集 S 中的记录.

$$\begin{cases}\Delta_1(I_t^i)=\{I_t^i[j]\,|\,|S_{I_t^i}[j]|\geq\min(|s|*\theta),\\ \qquad\qquad s\in B_i, I_t^i[j]\in I_t^i\}\\ \Delta_2(I_t^i)=\{I_t^i[j]\,|\,|S_{I_t^i}[j]|\leq\max(|s|/\theta),\\ \qquad\qquad s\in B_i, I_t^i[j]\in I_t^i\}\end{cases}\tag{3}$$

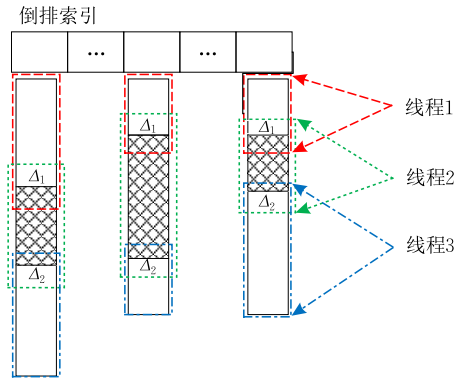


图 4 共享索引的扫描范围

按照上述的方式为每个线程分配数据块及相应的索引扫描范围，线程在扫描索引获取候选对的过程中不会丢失可能的结果. 如果获取的候选记录对来自不同的块中，由于数据在内存且是共享的，在验证的过程中也不会丢失结果.

由于数据是按照长度递增的顺序组织的，在算法实现的过程中，可以利用数据长度有序的特征，进一步减少每一个线程索引的扫描范围. 每一个线程在扫描索引的过程中，只需要扫描 I_t^i 及其扩展部分 $\Delta_1(I_t^i)$ 即可保证获取所有结果，可以避免重复计算和重复的中间结果.

算法 1. PSJoin_SharedIndex(S,θ,N).

输入：数据集 S，阈值 θ，线程个数 N

输出：相似度不小于 θ 的数据对

```
1.  //数据划分
2.  FOR i=0→N DO
3.      B_i←DataPartition(i);
4.  //索引构建
5.  FOREACH s in B_i DO {
6.      T_s^p←get prefix tokens of s;
7.      FOREACH t∈T_s^p DO
8.          I_t^i.add(t);
9.      }
10. //任务分解与执行
11. FOR i=0→N DO
12.     FOREACH t in U DO {
13.         I_t^i.range←{Δ1(I_t^i)∪I_t^i};
14.     Thread_Execution(&JoinThread1, &i);
15. }
```

共享索引的并行相似连接方法的实现细节如算法 1 PSJoin_SharedIndex 所示. 首先，主进程根据用

户指定的执行并行任务的线程个数 N，将输入数据集 S 划分成 N 块不相交的数据块(1～3 行). 然后，主进程利用每一个数据块 B_i 构建共享的倒排索引，对 B_i 中的每一条数据 s 获取其前缀 T_s^p，并将 T_s^p 中的每一个元素 t 加入到对应的倒排列表 I_t^i(4～8 行). 最后，主进程为每个线程分配实际需要扫描的每一个倒排列表的范围 I_t^i.range(11～13 行)，并启动 N 个线程执行 JoinThread1. 每一个线程利用主进程分配的数据块和需要扫描的索引范围并行地执行相似连接任务，如算法 2 所示，具体的实现细节参见 PPJoin+[15].

算法 2.　JoinThread1(i).

输入：线程编号

输出：相似记录

```
1.  FOREACH s in B_i DO {
2.    FOREACH t in T_s^p DO {
3.      tmp=ProbeIndex(I_t^i.range);
4.      Candidates.add(tmp);
5.    }
6.  Verification();
7.  OutputResults();
8.  }
```

5.2　独立索引的并行相似连接

基于独立索引的并行相似连接方法，进程负责数据的划分，多线程利用分配的数据块负责构建独立的局部索引，并执行相似连接任务. 由于相似的数据可能分布在相邻数据块中，再加上线程构建的独立索引在线程的子空间中，不能为其它线程直接访问. 因此，不能采用如图 4 中的方法来解决数据划分可能导致的结果丢失问题. 针对这个问题，在数据划分的时候为每一个数据块 B_i 按照相似记录的长度约束增加两部分配额 δ_1 和 δ_2，如图 5 所示.

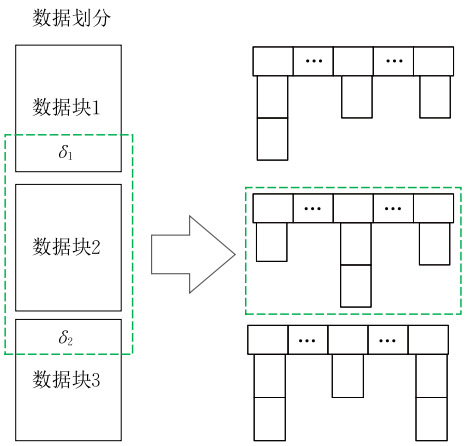


图 5　独立索引构建

$$\begin{cases}\delta_1(B_i)=\{s_j\mid |s_j|\geq\min(|s_k|*\theta), s_k\in B_i, s_j\in B_{i-1}\}\\ \delta_2(B_i)=\{s_j\mid |s_j|\leq\max(|s_k|/\theta), s_k\in B_i, s_j\in B_{i+1}\}\end{cases} \tag{4}$$

每个线程根据进程分配的数据块 B_i 计算自己的两部分配额 δ_1 和 δ_2，与数据块 B_i 中记录相似的记录肯定在 $\delta_1(B_i)\cup B_i\cup\delta_2(B_i)$ 中. 如果利用数据块 B_i 以及 δ_1 和 δ_2 来构建独立的索引，在执行相似连接的过程中线程只需要处理数据块 B_i 中的数据，利用 B_i 中的每一条数据的前缀去扫描线程子空间内的独立索引即可保证不丢失结果. 如果在扫描索引时获取的候选数据来自 δ_1 和 δ_2，需要在最后验证的阶段去访问进程空间来获取数据.

由于相邻的数据块之间的配额是有重复的，利用 $\delta_1(B_i)\cup B_i\cup\delta_2(B_i)$ 构建的索引之间也会有重复. 因此，在执行相似连接的过程中会有重复计算并产生重复的中间结果. 为了避免以上情况的产生，可以利用数据集有序的特征，利用 $\delta_1(B_i)\cup B_i$ 为相应的数据块构建索引，在执行相似连接的过程中扫描此索引即可得到全部结果.

算法 3.　PSJoin_IndependentIndex(S,θ,N).

输入：数据集 S，阈值 θ，线程个数 N

输出：相似度不小于 θ 的数据对

```
1.  //数据划分
2.  FOR i=0→N DO {
3.    B_i←DataPartition(i);
4.    B_i.range←{δ1(B_i)∪B_i}; }
5.  //任务分解与执行
6.  FOR i=0→N DO
7.    Thread_Execution(&JoinThread2,&i);
```

独立索引的并行相似连接方法的实现细节如算法 3 PSJoin_IndependentIndex 和算法 4 JoinThread2 所示. 其中算法 3 是由主进程需要完成的任务. 根据用户指定的线程个数 N，将数据划分成 N 块并为每一数据块 B_i 分配配额 δ_1 和 δ_2，确定每一个线程构建独立索引需要扫描的数据范围 B_i.range(2～4 行). 然后，启动 N 个线程并行的执行算法 4 中的任务(6～7 行). 在算法 4 中，每一个线程对其数据 B_i.range 内的每一条数据 s，获取其前缀 T_s^p，并将 T_s^p 中的每一个元素 t 加入到倒排列表 I_t^i(2～5 行). 线程在执行相似连接任务的过程中只需要处理数据块 B_i 中的数据(8 行). 对于数据块 B_i 中的每一条数据 s 获取其前缀 T_s^p，利用 T_s^p 中的元素去扫描线程子空间内的独立索引得到候选记录对(9～11 行). 最后，对得到的候选记录进行验证，得到所有相似度不低于

θ 的记录对.

算法 4. JoinThread2(i).

```
1. //索引构建
2. FOREACH s in B_i.range DO {
3.    T_s^p ← get prefix tokens of s;
4.    FOREACH t in T_s^p DO
5.       I_i^i.add(t);
6.  }
7. //执行相似连接
8  FOREACH s in B_i DO {
9.    FOREACH t in T_s^p DO {
10.      tmp=ProbeIndex(I_i^i);
11.      Candidates.add(tmp);
12.      Verification();
13.      OutputResults();
14.   }
15. }
```

为了避免线程对倒排列表的重复扫描,独立索引中的每一个倒排列表都有一个指针,该指针指向倒排列表的开始位置.随着执行过程的进行(集合的长度逐渐变长),该指针会根据记录的长度变化逐渐向倒排列表的尾部移动.如果线程当前处理的记录长度为 L,那么线程扫描倒排列表的开始位置,即指针应该指向倒排列表中第一个长度为$\lceil L*\theta \rceil$的记录所在的位置.也就是说,对于长度为 L 的数据,在倒排列表扫描的过程中从第一个长度为$\lceil L*\theta \rceil$的记录所在的位置开始,到当前记录所在的位置结束.这样保证了同一个倒排列表中的记录,在执行的过程中只会相遇一次.

5.3 算法分析

根据前面的论述可知,两种不同的数据划分方法和两种不同的任务分解方法可以组合成 4 种不同的并行相似连接的实现方式.本节选择其中的两种典型的实现方式进行分析:(1)数据量均衡划分与共享索引的实现方式;(2)数据长度均衡划分与独立索引的实现方式.在分析的过程中,包含索引构建和相似连接执行两部分的时间代价.假设数据集中数据量为 $Q=|S|$,N 为线程个数,c_{ib},c_{ip},c_{it} 分别为索引更新、扫描以及集合间交运算所需要的单位时间,α,β,γ 分别以上 3 种运算与数据长度的比例参数.

(1)数据量均衡划分与共享索引的实现方式

索引的构建由进程完成,构建索引的代价与数据量及数据长度成正比,索引构建的代价为

$$C_{IB}^{\text{EQ-SI}}=\sum_{k=0}^{Q}|s_k|\times\alpha\times c_{ib}.$$

相似连接的执行过程为 N 个线程并行地执行,算法执行的时间由执行时间最长的线程决定.执行的过程包含两部分,一部分是扫描倒排索引确定候选集,另一部分是当前数据与候选集合中的数据进行验证.由于数据集经过预处理以后是按照长度递增的顺序排放的,并且每个数据块中的数据量是相同的,那么执行时间最长的线程是处理最后一个数据块 B_{N-1} 的线程,它扫描倒排索引确定候选集的时间为

$$\begin{aligned}C_{IP}^{\text{EQ-SI}}&=L_{B_{N-1}}\times\beta\times c_{ip}\\&=\sum_{j=Q-Q/N}^{Q}|s_j|\times\beta\times c_{ip}.\end{aligned}$$

其中 $L_{B_{N-1}}$ 为最后一个数据块中数据的总长度.B_{N-1} 中的数据量为 Q/N,由于最后的数据块 B_{N-1} 中包含的都是数据集中较长的数据,那么 $L_{B_{N-1}}>\sum_{k=0}^{Q}|s_k|/N$.

索引扫描的过程中会确定当前数据的候选集,在不考虑使用过滤方法的情况下,扫描索引的时间决定了候选集的大小.所以,当前数据与候选集中的数据做集合交运算的代价与扫描索引的代价成正比,与当前数据的长度成正比.那么,当前数据与候选集合验证的时间代价为

$$\begin{aligned}C_{IT}^{\text{EQ-SI}}&=\sum_{j=Q-Q/N}^{Q}|s_j|\times\beta\times c_{ip}\times|s_j|\times\gamma\times c_{it}\\&=\beta\times c_{ip}\times\gamma\times c_{it}\sum_{j=Q-Q/N}^{Q}|s_j|^2,\end{aligned}$$

那么,此实现方式需要的总时间为

$$C^{\text{EQ-SI}}=C_{IB}^{\text{EQ-SI}}+C_{IP}^{\text{EQ-SI}}+C_{IT}^{\text{EQ-SI}}.$$

(2)数据长度均衡划分与独立索引的实现方式

数据长度均衡划分的方法产生的数据块中数据的总长度都相同,为$\sum_{k=0}^{Q}|s_k|/N$.那么,每个线程根据自己分配的数据块 B_i 及配额 δ_1 和 δ_2 构建独立的索引,这里我们认为 δ_1 和 δ_2 包含的数据量远小于 B_i.构建索引时间为

$$C_{IB}^{\text{EL-DI}}=\left(\sum_{k=0}^{Q}|s_k|/N\right)\times\alpha\times c_{ib}.$$

线程扫描倒排索引获取候选集的时间为

$$C_{IP}^{\text{EL-DI}}=\left(\sum_{k=0}^{Q}|s_k|/N\right)\times\beta\times c_{ip}.$$

假设某个线程处理的数据块 B_i 中含有的数据量

为 Q'，数据的起始位置为 m.

那么，$C_{IP}^{\text{EL-DI}}=\sum_{j=m}^{m+Q'}|s_j|\times\beta\times c_{ip}$.

线程执行当前数据与候选集合中数据的集合交运算的时间为

$$C_{IT}^{\text{EL-DI}}=\sum_{j=m}^{m+Q'}|s_j|\times\beta\times c_{ip}\times|s_j|\times\gamma\times c_{it}$$

$$=\beta\times c_{ip}\times\gamma\times c_{it}\sum_{j=m}^{m+Q'}|s_j|^2.$$

那么，此实现方式需要的总时间为

$$C^{\text{EL-DI}}=C_{IB}^{\text{EL-DI}}+C_{IP}^{\text{EL-DI}}+C_{IT}^{\text{EL-DI}}.$$

通过上面的分析可以很明显地得出，$C_{IB}^{\text{EQ-SI}}>C_{IB}^{\text{EL-DI}}$，$C_{E}^{\text{EQ-SI}}>C_{E}^{\text{EL-DI}}$.

对于 $C_{IT}^{\text{EQ-SI}}$，由于数据集是按照长度递增的次序排放的，使用数据量均衡划分的方法对数据集进行划分，最后一个数据块中数据的总长度肯定大于按照长度均衡的方法得到的每个数据块的平均长度. 那么，

$$\sum_{j=Q-Q/N}^{Q}|s_j|^2>\frac{Q}{N}\left(\sum_{k=0}^{Q}|s_k|/N\right)^2$$

$$=\frac{1}{N^2}\times\frac{Q}{N}\left(\sum_{k=0}^{Q}|s_k|\right)^2.$$

对于 $C_{IT}^{\text{EL-DI}}$，由于此实现方式中数据按照等长度划分，每一个数据块 B_i 中数据的总长度相同，设 B_i 中数据为从序号 m 开始的 Q' 个. 即

$$\sum_{j=m}^{m+Q'}|s_j|=\sum_{k=0}^{Q}|s_k|/N,\ \sum_{j=m}^{m+Q'}|s_j|^2<\frac{1}{N^2}\left(\sum_{k=0}^{Q}|s_k|\right)^2.$$

因此，$C_{IT}^{\text{EQ-SI}}>C_{IT}^{\text{EL-DI}}$.

综上所述可得，$C^{\text{EQ-SI}}>C^{\text{EL-DI}}$，数据长度均衡划分与独立索引的实现方式会优于数据量均衡划分与共享索引的实现方式.

5.4　R-S 并行相似连接

前面两节论述的是有关数据集执行自连接的实现方式及时间代价分析. 本节论述两个不同的数据集之间执行 R-S 并行相似连接的实现方式. 与自连接不同的是，两个不同的数据集之间的划分要满足不同划分块之间要保证长度约束来保证不丢失可能的结果. 因此，与关系数据库中的 R-S 连接类似需要选定内表和外表. 然后，将选定一个数据集 R 按照实现的方式进行数据划分，数据集 R 的划分结果决定数据集 S 的划分.

如果采用基于共享索引的方式实现并行的相似连接，首先选择数据集 R，数据划分方式与前面论述的方式相同；然后，利用数据集 S 构建共享的索引；最后，为 R 的每一个数据块确定其扫描索引的范围.

如果采用基于独立索引的方式实现并行相似连接，需要对两个数据集进行划分，两个数据集之间的划分要满足一定的关系. 数据集 R 和 S 的划分个数相同，数据集 R 的数据块 B_i^R 与数据集 S 对应的数据块 B_i^S 之间必须满足以下关系才能保证不丢失结果.

$$\begin{cases}\max(|s_j|)\leq\max(|s_k|/\theta)\\ \min(|s_j|)\geq\min(|s_k|\times\theta)\\ s_j\in B_i^S,\ s_k\in B_i^R\end{cases}\tag{5}$$

具体的实现细节与算法 4 和算法 5 类似. 不同的地方是，在数据划分的过程中需要根据数据集 R 的划分结果去决定数据集 S 的划分；在索引构建的过程中使用的是数据集 S 的数据块 B_i^S；在执行相似连接的过程中扫描的数据块是 B_i^R.

6　实　验

为了验证本文提出的基于多核的并行相似连接方法的性能和可扩展性，以 PPJoin+ 的算法核心为基础在多核多处理器平台上利用不同的实验条件进行了详细的实验及分析，包括数据划分方法、任务划分方法、线程个数及处理器核数对相似连接效率的影响，不同实验条件下的加速比以及可扩展性. 在实验分析中，基于本文提出的数据划分和任务分解方法实现了 4 种不同的并行相似连接算法：数据量均衡划分与共享索引的方法（EQ-SI），数据长度均衡划分与共享索引的方法（EL-SI），数据量均衡划分与独立索引的方法（EQ-DI），数据长度均衡划分与独立索引的方法（EL-DI）.

6.1　实验条件

本实验所使用的多核多处理器平台的配置为：Intel Xeon 4×4 cores 2.4GHz，48GB 内存，操作系统 Ubuntu Server 14.04；验证处理器核心数量对并行相似连接性能影响的实验部分租用 Amazon EC2 的服务器. 实验中所使用的程序用 C/C++ 编写，多线程的实现使用了 POSIX Thread，编译器版本 GCC 4.8.4，编译程序使用了-O3 选项. 实验的过程中线程的个数与数据划分的数据块的数量相同. 实验中所使用的 3 个数据集 CiteSeer、DBLP 和 DataWB. 其中 CiteSeer 和 DBLP 均为文献出版信息，CiteSeer 包含 467900 条记录，最小记录长度为 1，最大记录长度为 113，平均记录长度 10；DBLP 包含 1021062

条记录,最小记录长度为 2,最大记录长度为 44,平均记录长度 6.8. DataWB 数据集是用抓取工具从某知名微博网站抓取获得,我们从中抽取了 300 万条用户发表的信息,最小记录长度为 1,最大记录长度为 109,平均记录长度为 11.3.

6.2 实验结果与分析

(1)数据划分和任务分解对性能的影响

本实验对提出的数据划分和任务分解方法对并行相似连接性能的影响进行了实验分析. 实验使用了 DBLP 和 DataWB 两个数据集和 Jaccard 相似度计算方法对 4 种不同的实现方式进行了测试,线程个数 16,实验的结果如图 6 和图 7 所示. 从图中的结果可以看出,并行相似连接算法的 4 种不同的实现方式在同样的数据集上完成同样的工作所需要的时间远低于单进程的 PPJoin+[15]和 AdaptJoin[17]. 随着相似度阈值的降低,每一条数据的前缀个数增加,在整个数据集上执行相似连接的代价增加,使用并行的处理方式可以提高算法的执行效率. 从图 6 显示的结果可以看出,在 DBLP 数据上,当 $\theta=0.9$ 时,EQ-SI 的性能与 PPJoin+和 AdaptJoin 相比分别提高了 61%和 30%,EL-DI 的性能与 PPJoin+和 AdaptJoin 相比分别提高了 69%和 61%;当 $\theta=0.7$ 时,EQ-SI 的性能与 PPJoin+和 AdaptJoin 相比分别提高了 70%和 46%,EL-DI 的性能与 PPJoin+和 AdaptJoin 相比分别提高了 80%和 76%. 在 DataWB 数据集上,当 $\theta=0.9$ 时,EQ-SI 的性能比 PPJoin+提高了 68%,EL-DI 的性能比 PPJoin+提高了 94%;当 $\theta=0.7$ 时,EQ-SI 的性能比 PPJoin+提高了 64.9%,EL-DI 的性能比 PPJoin+提高了 85%. 分析结果可以得出,EL-DI 并行实现方式的优势更加明显.

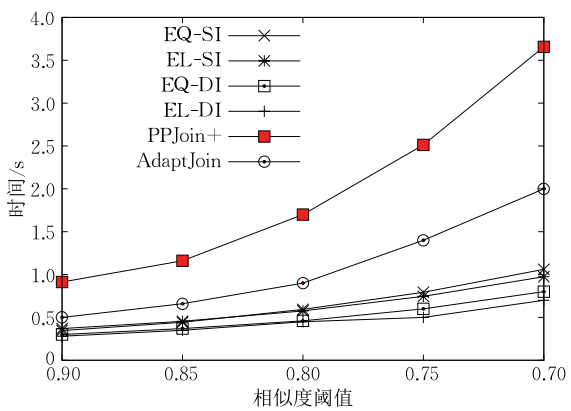


图 6 不同实现方法的比较(DBLP)

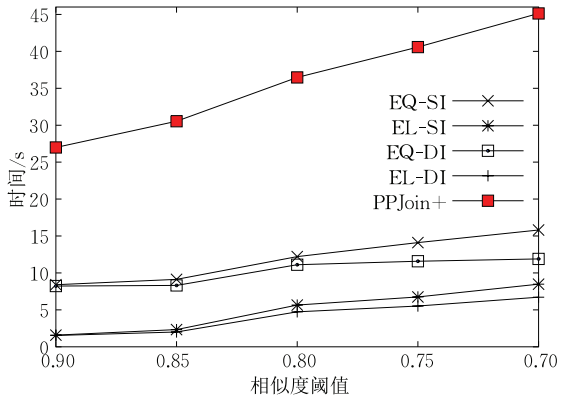


图 7 不同实现方法的比较(DataWB)

对于数据划分方法,当算法使用相同的任务划分方法时,等长数据划分比等量数据划分更能提高相似连接的效率. 如图 6 和图 7 中所示,EL-SI 的性能高于 EQ-SI 的性能 2%~8%(DBLP)和 46%~80%(DataWB),EL-DI 的性能高于 EQ-DI 的性能 1.2%~16%(DBLP)和 43%~81%(DataWB). 形成以上结果的原因在于相似连接操作的复杂度与数据长度成正比,数据长度越大,数据的前缀就越多,索引扫描及过滤方法的代价就越大. 如果采用等数据量划分的方法,会导致不同的处理器核心上分配的任务不均衡,最终造成算法整体性能的下降. 因此,对于相似连接算法来说,采用等长数据划分能更好地实现不同处理器核心上的任务均衡,实现算法整体性能的提高.

对于任务分解方法,当算法使用相同的数据划分方法时,独立索引的任务划分方法比共享索引的方法更有效. 从图 6 和图 7 中可以看出,EL-DI 的性能优于 EL-SI 21%~28%(DBLP)和 2%~20%(DataWB),EQ-DI 的性能优于 EQ-SI 13%~24%(DBLP)和 2%~24.7%(DataWB),并且 EQ-DI 与 EL-DI 的性能都高于 EQ-SI 与 EL-SI. 形成以上结果的原因在于,基于独立索引的实现方式使得算法的并行度进一步提高. 在 EQ-DI 与 EL-DI 的实现中,主进程只负责数据的划分,多个线程在自己的线程空间并行地构建独立的索引,并行地执行相似连接任务. 在 EQ-SI 与 EL-SI 实现中,主进程负责数据的划分和共享索引的构建,多个线程负责并行的执行相似连接任务. 此实现方式并行化程度不够高,而且由于多个线程使用共享的索引导致 cache 的不一致性增加.

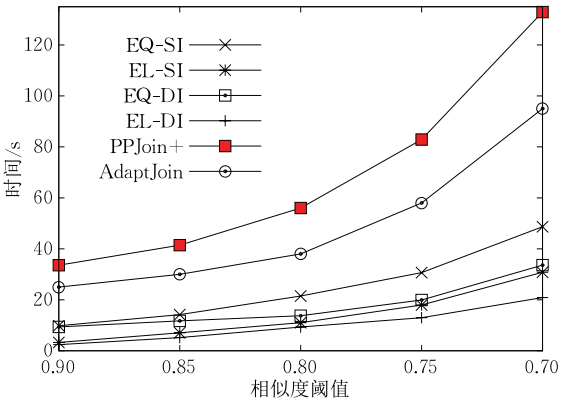


图 8　Cosine 相似度函数(DataWB)

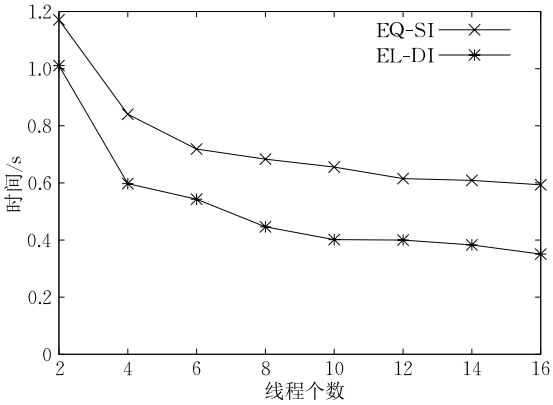


图 9　线程个数对性能的影响(DBLP,Jaccard)

(2)其它相似度函数

为了验证本文的并行相似连接算法对于其它相似度函数的适用性,本实验在 DataWB 数据集上使用 Cosine 方法,对本文中 4 种不同的实现方式与 PPJoin+和 AdaptJoin 进行了对比测试,实验中线程个数设置为 16. 测试结果如图 8 所示. 从图 8 中的显示可以看出,EQ-SI 和 EL-DI 两种不同的并行相似连接实现方法当使用 Cosine 相似度计算方法时也非常有效. 当 $\theta=0.9$ 时,EQ-SI 的性能为 PPJoin+的 3.4 倍、AdaptJoin 的 2.5 倍;EL-DI 的性能为 PPJoin+的 13.8 倍、AdaptJoin 的 10 倍;当 $\theta=0.7$ 时,EQ-SI 的性能为 PPJoin+的 2.7 倍、AdaptJoin 的 1.9 倍;EL-DI 的性能为 PPJoin+的 14 倍、AdaptJoin 的 10 倍.

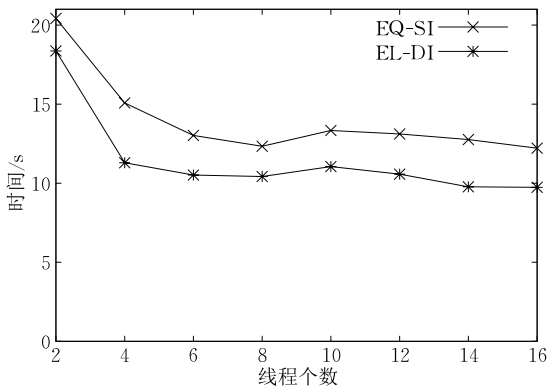


图 10　线程个数对性能的影响(DataWB,Jaccard)

(3)并行线程个数对性能的影响

本实验使用了 DBLP 和 DataWB 两个数据集,对线程个数对并行相似连接算法的影响进行了测试. 实验中,相似度阈值 $\theta=0.8$,使用 Jaccard 和 Cosine 两种相似度计算方法,在同一个服务器上不同线程个数的情况下对 EQ-SI 和 EL-DI 两种不同的实现方法的性能进行了测试. 为了防止任务调度系统对线程的影响,对线程设置了 CPU 亲和性参数,保证线程在同一个处理器核心上执行. 实验结果如图 9、图 10 和图 11 所示.

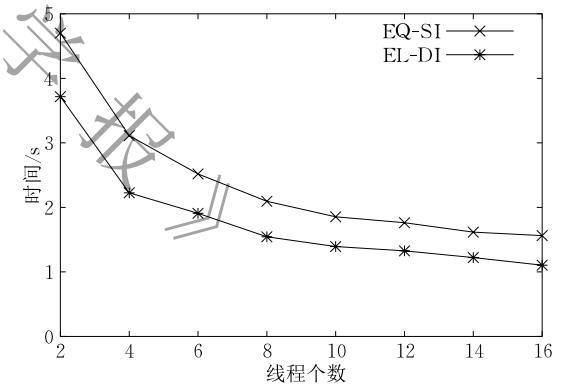


图 11　线程个数对性能的影响(DBLP,Cosine)

图 9 和图 10 显示的是在 DBLP 和 DataWB 不同的数据集上使用相同的相似度计算方法 Jaccard 的实验结果. 可以看出,随着线程个数的增加,完成同样的并行连接任务 EQ-SI 和 EL-DI 所需要的时间减少. 在图 10 中,当线程的个数为 2 时 EQ-SI 运行的时间为 20.4 s,当线程的个数为 16 时 EQ-SI 的运行时间为 12.2 s,性能提高了 40%;同样的条件下,EL-DI 使用 16 个线程时的性能比使用 2 个线程时的性能提高了 46%. 本实验中使用的系统处理器的配置为 4 个 4 核的 CPU,随着线程个数的增加,每个 CPU 上运行的线程增多,由于同一个 CPU 上的核心之间共享一级或者二级缓存,因此 cache 的竞争也随之增加,性能提高的比例也随之降低.

图 9 和图 11 显示的是在 DBLP 数据集上使用 Jaccard 和 Cosine 两种不同相似度计算方法的实验结果. 由于同样的相似度阈值的情况下,使用 Cosine 时同一个数据的前缀比使用 Jaccard 时多,因此线程个数相同的情况下,Cosine 方法会消耗更多的时间. 图 11 中,当线程个数为 2 时 EQ-SI 的运行时间为 4.7 s, 当线程个数为 16 时 EQ-SI 的运行

时间为 1.56 s,性能提高了 66.7%;同样的条件下,EL-DI 的性能提高了 70%.

(4)处理器核数对性能的影响

在本实验中,我们验证处理器核数对并行相似连接性能的影响,实验的结果如图 12 所示.为了实现实验所需要系统 CPU 核数的变化,本实验使用了 Amazon EC2 的服务,通过系统配置生成具有不同 CPU 核数的服务器,在本实验中使用的系统 CPU 配置分别为 1×4 Cores CPU,2×4 Cores CPU,4×4 Cores CPU,8×4 Cores CPU,其中每个核心均为 2.4 GHz, Intel Xeon E5-2676v3.实验中使用的数据集为 DataWB、Jaccard 相似度计算方法、$\theta=0.8$,每个系统上运行的线程数均设为 16.

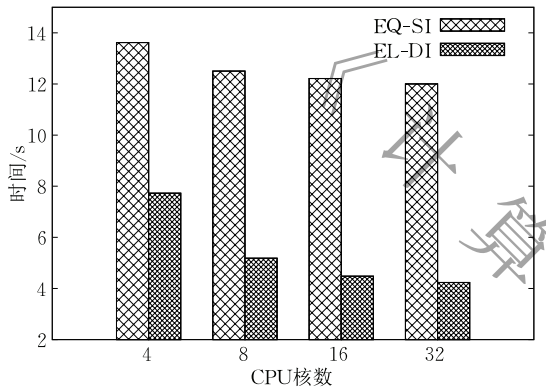


图 12 CPU 核心个数对性能的影响

从图 12 显示的实验结果来看,随着处理器核数的增加 EQ-SI 和 EL-DI 两种实现方法执行同样的任务消耗的时间在不断地降低,即并行相似连接的性能在不断地提高.随着处理器核数的增加,线程之间的资源竞争减少,所以时间消耗降低的幅度也随之减少.由于并行相似连接的线程个数均被设为 16,当处理器核数为 4 时,每个处理器核上平均有 4 个线程在运行,线程之间的资源竞争最大,因此消耗的时间最多.由于 EL-DI 的索引是在线程的空间,线程的调度会造成线程之间的 cache 和内存竞争比较频繁,所以处理器核数的增加对 EL-DI 的性能影响较大.当处理器核数为 32 时,16 个线程会被调度到不同的处理器核心上执行,线程之间的资源竞争减少.所以,当处理器核心数超过并行运行的线程个数时,处理器核心核数的增加对并行相似连接性能的影响很少.

(5)并行相似连接的加速比

图 13 显示的是线程个数与并行相似连接加速之间的关系.本实验使用两个数据集 DBLP 和 DataWB 对性能表现最好的并行相似连接实现方法 EL-DI 的加速比进行了测试,使用 Jaccard 相似度计算方法,相似度阈值设为 0.8,处理器配置为 4 个 4 核 CPU.加速比的计算方法为 PPJoin+单进程执行时间与 EL-DI 的并行执行时间之间的比值.

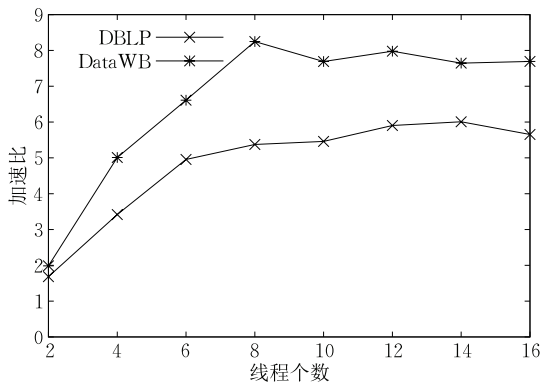


图 13 加速比(EL-DI,Threads)

从图 13 中的结果显示,随着线程个数的增加 EL-DI 的加速比也随之提高.当线程个数为 2 时,在两个数据集上 EL-DI 的加速比为 2 左右.对于 DataWB 数据集,当线程个数为 8 时,EL-DI 的加速比达到最大值接近 8.在 DBLP 数据集上,算法运行的总时间比较少(如图 6)并且随着线程个数的增加而减少,而结果的输出时间是基本固定的且占总时间的比例较大,所以计算出的加速比较低.从图 13 中 EL-DI 的加速比的变化曲线可以看出,随着使用的线程个数的增加,EL-DI 的加速比提高的趋势变小,并出现了抖动.造成这种现象的原因在于,随着并行运行的线程个数的增加,每个处理器上运行的线程数量增加,线程之间对内存和 cache 的竞争增多.

图 14 显示的是处理核数与加速比之间的关系.为了验证处理器核心数量对并行相似连接执行效率的影响,我们在 Amazon EC2 上通过系统配置生成具有不同处理器核数的服务器,具体参数同实验 4.本实验使用 DataWB 数据集和 Jaccard 相似度计算方法,相似度阈值设为 0.8,并行运行的线程个数与处理器核数相同,实验的算法为 EL-DI.从图 14 中可以看出,随着使用的处理器核心数量的增加以及并行线程个数的增加,加速比近似线性增加.在本文的实验中结果的输出是由主进程完成的,结果输出占用的时间基本固定.随着线程个数的增加,并行相似连接执行的过程所花费的时间减少,导致结果输出占用主要比例的时间,所以加速比的变化减小.

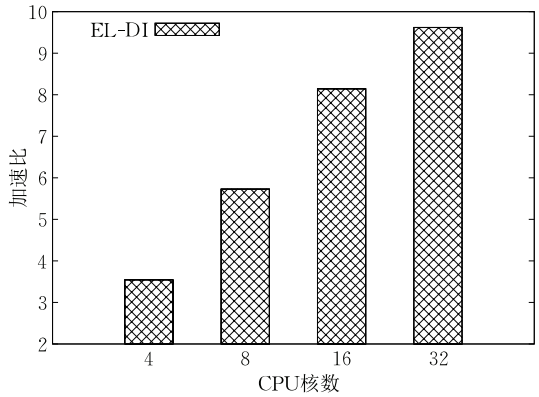


图 14　加速比(EL-DI,Cores)

(6) R-S 并行相似连接

本实验利用 CiteSeer 和 DBLP 两个数据集验证本文提出的 R-S 并行相似连接的执行效率. 实验系统的处理器为 4×4 Cores CPU,并行执行的线程数量为 16,使用 Jaccrad 相似度计算方法. 实验的结果如图 15 所示,其中 RS-EQ-SI 为基于数据量均衡划分与共享索引的方式实现的 R-S 并行相似连接;RS-EL-DI 为基于数据长度均衡划分与独立索引的方式实现的 R-S 并行相似连接. 在实验中,我们选择数据量小的 CiteSeer 来作为主划分,根据其划分的结果去划分 DBLP 数据集并构建独立索引. 图 15 的结果显示,随着相似阈值的提高 RS-EQ-SI 和 RS-EL-DI 运行的时间减少,这主要是由于随着相似度阈值的提高每条数据操作的代价减少所致. 从整体看 RS-EL-DI 的效率比 RS-EQ-SI 提高 10%～50%. RS-EQ-SI 和 RS-EL-DI 两种实现方式在 CiteSeer 和 DBLP 两个数据集上执行 R-S 并行相似连接操作均可以在 1 秒内完成.

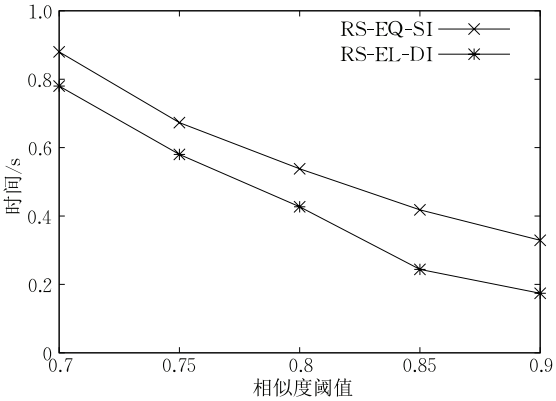


图 15　R-S 并行相似连接

7　结　论

本文为了提高相似连接的效率和计算资源的利用率,结合现代多核多处理平台的并行计算特点,提出了基于多核的细粒度并行的集合相似连接算法. 为了提高算法的并行性,对数据划分方法和任务分解方法进行了研究,提出了等量数据划分和等长数据划分方法,以及共享索引和独立索引的任务分解方法. 利用本文提出的数据划分和任务分解方法,实现了并行的相似自连接和并行的 R-S 相似连接. 实验结果表明,对于相似连接算法,采用等长数据划分和独立索引的并行化实现方式能最大程度地提高相似连接的效率. 由于现代的处理器都采用了多级 cache,在本文研究的过程中发现多级 cache 对并行算法的性能有很大的影响,我们将如何更好地利用多级 cache 作为下一步的研究工作.

参　考　文　献

[1] Albutiu M C, Kemper A, Neumann T. Massively parallel sort-merge joins in main memory multi-core database systems //Proceedings of the 38th International Conference on Very Large Data Bases. Istanbul, Turkey, 2012: 1064-1075

[2] Balkesen C, Alonso G, Teubner J, Özsu M T. Multi-core, main-memory joins: Sort vs. hash revisited//Proceedings of the 40th International Conference on Very Large Data Bases. Hangzhou, China, 2014: 85-96

[3] Balkesen C, Teubner J, Alonso G, Özsu M T. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware//Proceedings of the 29th IEEE International Conference on Data Engineering. Brisbane, Australia, 2013: 362-373

[4] Yu M, Li G, Deng D, Feng J. String similarity search and join: A survey. Frontiers of Computer Science, 2015, 11(24): 1-19

[5] Sebastian W, Deng D, Stefan G, et al. State-of-the-art in string similarity search and join. SIGMOD Record, 2014, 43(1): 64-76

[6] Lee K, Lee Y, Choi H, et al. Parallel data processing with MapReduce: A survey. SIGMOD Record, 2011, 40(4): 11-20

[7] Hernández M, Stolfo S. The merge/purge problem for large databases//Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. San Jose, USA, 1995: 127-138

[8] Dong X, Halevy A, Madhavan J. Reference reconciliation in complex information spaces//Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. Baltimore, Maryland, 2005: 85-96

[9] Sivic J, Zisserman A. Video Google: A text retrieval approach to object matching in videos//Proceedings of the 9th IEEE International Conference on Computer Vision. Nice, France, 2003: 1470-1477

[10] Bilenko M, Mooney R J. On evaluation and training-set construction for duplicate detection//Proceedings of the International Workshop on Data Cleaning, Record Linkage and Object Consolidation, Washington, USA, 2003: 7-12

[11] Fellegi I, Sunter A. A theory for record linkage. Journal of the American Statistical Association, 1969, 64(328): 1183-1210

[12] Christen P. A survey of indexing techniques for scalable record linkage and deduplication. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(9): 1537-1555

[13] Arasu A, Ganti V, Kaushik R. Efficient exact set-similarity joins//Proceedings of the 32nd International Conference on Very Large Data Bases. Seoul, Korea, 2006: 918-929

[14] Bayardo R, Ma Y, Srikant R. Scaling up all pairs similarity search//Proceedings of the 17th International Conference on World Wide Web. Alberta, Canada, 2007: 131-140

[15] Xiao C, Wang W, Lin X, Yu J. Efficient similarity joins for near duplicate detection//Proceedings of the 17th International Conference on World Wide Web. Beijing, China, 2008: 131-140

[16] Sarawagi S, Kirpal A. Efficient set joins on similarity predicates //Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. Paris, France, 2004: 743-754

[17] Wang J, Li G, Feng J. Can we beat the prefix filtering? An adaptive framework for similarity join and search//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. Scottsdale, USA, 2012: 85-96

[18] Xiao C, Wang W, Lin X. Ed-join: An efficient algorithm for similarity joins with edit distance constraints//Proceedings of the 34th International Conference on Very Large Data Bases. Auckland, New Zealand, 2008: 933-944

[19] Qin J, Wang W, Lu Y, et al. Efficient exact edit similarity query processing with the asymmetric signature scheme//Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. Athens, Greece, 2011: 1033-1044

[20] Rong Chuitian, Xu Tianren, Du Xiaoyong. Partition-based set similarity join. Computer Research and Development, 2012, 49(10): 2066-2076

[21] Rong Chuitian, Lu Wei, Wang Xiaoli, et al. Efficient and scalable processing of string similarity join. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(10): 2217-2230

[22] Li Guoliang, He Jian, Deng Dong, Li Jian. Efficient similarity join and search on multi-attribute data//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne, Australia, 2015: 1137-1151

[23] Jiang Yu, Deng Dong, Wang Jiannan, et al. Efficient parallel partition-based algorithms for similarity search and join with edit distance constraints//Proceedings of the Joint EDBT/ICDT 2013 Workshops. Genoa, Italy, 2013: 18-22

[24] Li Guoliang, Deng Dong, Wang Jiannan, Feng Jianhua. PASS-JOIN: A partition-based method for similarity joins//Proceedings of the 37th International Conference on Very Large Data Bases. Seattle, USA, 2011: 253-264

[25] Lieberman M D, Sankaranarayanan J, Samet H. A fast similarity join algorithm using graphics processing units//Proceedings of the 24th IEEE International Conference on Data Engineering. Cancun, Mexico, 2008: 1111-1120

[26] Vernica R, Carey M, Li C. Efficient parallel set-similarity joins using MapReduce//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indianapolis, USA, 2010: 495-506

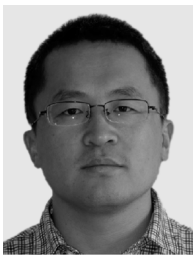
[27] Metwally A, Faloutsos C. V-SMART-Join: A scalable MapReduce framework for all-pair similarity joins of multisets and vectors//Proceedings of the 38th International Conference on Very Large Data Bases. Istanbul, Turkey, 2012: 704-715

[28] Deng D, Li G, Hao S, et al. MassJoin: A MapReduce-based algorithm for string similarity joins//Proceedings of the 29th IEEE International Conference on Data Engineering. Brisbane, Australia, 2013: 340-351

[29] Kolb L, Thor A, Rahm E. Load balancing for MapReduce-based entity resolution//Proceedings of the 28th IEEE International Conference on Data Engineering. Washington, USA, 2012: 618-629

[30] Teaching Materials Writing Group of Multi-Core. Multi-Core Programming. Beijing: Tsinghua University Press, 2007(in Chinese)
(多核系列教材编写组. 多核程序设计. 北京: 清华大学出版社, 2007)

[31] Asia-Pacific Research and Development Ltd. Multicore and Multithread Technology. Shanghai: Shanghai Jiaotong University Press, 2011(in Chinese)
(英特尔亚太研发有限公司. 多核多线程技术. 上海: 上海交通大学出版社, 2011)



RONG Chui-Tian, born in 1981, Ph. D., associate professor. His main research interests include database and information retrieval, cloud computing and big data analysis.

LI Yin-Yin, born in 1992, M. S. candidate. His research interests include cloud computing and big data analysis.

FENG Lin-Jing, born in 1991, M. S. candidate. Her research interests include database and information retrieval.

WANG Jian-Ming, born in 1974, Ph. D., professor. His main research interests include computer vision, pattern recognition and big data analysis.

Background

In the big data era, the main task of data management is to process the large scale and heterogeneous data from different data sources. During the process of data integration and data analysis, there is a need to join the data that refer to the same entity of the world. However, we cannot do that by using the join operation of the traditional database, as there are not unique keys for these data. So, the similarity join is proposed to join the data from different sources. Similarity join is a primitive operation that widely used in many applications. It is used to find all similarity pairs whose similarity are not less than the given threshold from two datasets using the given similarity functions.

As the development of information technologies, such as Internet and mobile applications *et al*., the scale of the data is increasing vastly. In order to analysis large scale data sets, there is a need to improve the performance of computers. So, the architecture of computer with multicores and multi processors has been invented to satisfy the increasing need of computation and to improve the computer's efficiency and performance.

To our best knowledge, all the existing similarity join works have focused on devising complicate algorithms. And, none of them has exploited the characteristics of multicores. Several works that focus on traditional database operations, such as hash join, have demonstrated that they can achieve high performance by exploiting the parallel computing on multicores. Inspired by their works, we proposed a solution for parallel similarity join based on multicores to improve its efficiency. By the fully understanding of characteristics of similarity join and the modern computer architecture, we proposed the methods for data partition and task decomposition to improve the parallelism of similarity join. The extensive experiments have been conducted under different conditions. The experimental results demonstrate that the solutions proposed in this work can exploit the parallel computation capacity of multicores efficiently, and then improve the efficiency of similarity join significantly.

This work was supported by the project of National Natural Science Foundation of China (No. 61402329, No. 61373104) and China Scholarship Council. There is a research agenda in these projects to explore the new framework to implement parallel similarity join.