基于数据流的k-近邻连接算法

王 飞 秦小麟 刘 亮 沈 尧
(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 k-近邻连接查询是空间数据库中一种常用的操作,该查询处理过程涉及连接和最近邻查询两个复杂操作。传统的集中式k-近邻连接查询算法已不能适应当前呈爆炸式增长的数据规模,设计分布式k-近邻连接查询算法成为了目前亟需解决的问题。现有的分布式k-近邻连接查询算法都包括了多轮串行的MapReduce任务,而每个MapReduce任务均需要读写分布式文件系统,导致MapReduce不能有效表达多个任务之间的依赖关系,因此算法效率低下。首先提出了一种基于数据流的计算框架,该框架建立在MapReduce之上,将数据处理过程按照数据流图建模。在该框架基础上,提出了一种高效的k-近邻连接算法,它利用空间填充曲线将多维数据映射为一维数据,从而将k-近邻连接查询转化为一维范围查询。实验结果表明,该算法的可扩展性较高,且效率比现有算法更优。

关键词 k-近邻连接,数据流,MapReduce,计算框架

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.5.041

Algorithm for k-Nearest Neighbor Join Based on Data Stream

WANG Fei QIN Xiao-lin LIU Liang SHEN Yao
(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract kNN join is a frequently used operation in spatial database. It involves both the join and the NN search. Data scale is exploding, and traditional centralized algorithm can not meet the requirements. It is an urgent problem to design distributed kNN join algorithm currently. Existing distributed algorithms include several rounds of serial MapReduce tasks, but each MapReduce task reads and writes data from distributed file system. It is inefficient when expressing dependencies between jobs, and these algorithms are inefficient. Firstly, we proposed a framework based on data stream on MapReduce. This framework models data handle process according to the data flow diagram, and we proposed an efficient kNN join algorithm on the framework. The algorithm maps multi-dimensional data sets into one dimension using space-filling curves (z-values), and transforms kNN joins into a sequence of one-dimensional range searches. Experimental results demonstrate that the algorithm can efficiently resolve the large scale kNN join spatial query.

Keywords kNN join, Data stream, MapReduce, Framework

1 引言

k-近邻连接(k-Nearest Neighbor Join,kNNJ)查询是空间数据库中一种非常重要和频繁的数据库操作,在k-近邻分类、k-均值聚类、抽样评估、遗漏值评估等应用中,kNNJ得到了广泛的应用[1-3]。由于近邻(NN)查询和join查询在数据库中都是极为耗时的操作,尤其当数据集很大或者维度很高时,kNN join的代价将会变得惊人。

随着空间数据急剧增长,单节点单进程算法[4-6]已不再适用,设计云计算环境下的分布式kNNJ查询算法成为了目前亟需解决的问题。MapReduce[7]是Google提出的一种集群上处理大规模数据集的分布式并行编程模型,也是目前云计算环境中的核心计算模式。为了有效地解决大规模数据集的kNNJ查询问题,文献[8-10]都提出了Hadoop平台下较高效的两个数据集之间的kNNJ算法。这些算法将kNN join查询转化成3个串行执行的MapReduce任务。在现有的Hadoop[11]平台下,每个MapReduce任务都需要读写HDFS,即使某些MapReduce任务产生的数据仅是临时数据。现有算法需要从HDFS分别读写3次,且HDFS默认写3份。这种表达作业依赖关系的方式是低效的,在读写数据量很大时会产生大量不必要的I/O开销,从而导致算法效率低下。

为了解决上述问题,本文提出了一种适用于多个具有依赖关系的MapReduce任务的计算框架——基于数据流的计算框架。该框架将数据处理过程不再按照单任务建模,而是作

到稿日期:2014-07-05 返修日期:2014-09-27 本文受国家自然科学基金项目(61373015,61300052),国家教育部高等学校博士学科点专项科研基金资助项目(20103218110017),江苏高校优势学科建设工程资助项目(PAPD),中央高校基本科研业务费专项项目(NP2013307),云计算-南航-大数据处理引擎技术研究项目资助。

王 飞(1989-),男,硕士生,主要研究方向为云环境下数据查询处理,E-mail:wangfnuaa@163.com;**秦小麟**(1953-),男,教授,博士生导师,主要研究方向为分布式数据管理与安全、信息安全等;**刘 亮**(1985-),男,博士后,讲师,主要研究方向为传感器网络数据库、流数据库等;**沈 尧**(1986-),男,博士生,主要研究方向为云计算。

为一种数据流图来处理。基于数据流的计算框架把MapReduce过程拆分成若干个子过程，同时可以把多个具有依赖关系的 MapReduce 任务组合成一个较大的 DAG 任务，减少了多个具有依赖关系 MapReduce 任务之间的文件存储。合理组合各个子过程，也可以减少任务的执行时间。

同时，在基于数据流的计算框架上提出了面向大规模空间数据集的 kNN Join 查询算法 D-kNNJ (Data Stream based kNN join)。该改进算法利用空间填充曲线将多维数据映射为一维数据，从而将 k-近邻连接查询转化为一系列的一维范围查询，包括数据划分和 kNN Join 查询两个步骤。其中数据划分需要解决两个问题：1)数据偏斜，数据的偏斜问题是影响分布式算法效率的关键因素之一；2)在满足查询要求的同时减少数据的复制，由于 kNN join 查询中数据的紧耦合性，将可能满足查询要求的数据都划分到同一分块中，并且尽可能减少数据的复制具有较大难度。

D-kNNJ 算法在解决这两个问题时采用了传统的采样和空间填充曲线相结合的方法，在采样基础上，对样本点按照空间填充曲线 z-value 进行排序，根据等数据量原则，确定数据的划分。在分块中进行 kNN join 查询时，D-kNNJ 算法使用 B 树来加快查询的效率。由于 D-kNNJ 算法只需要从 HDFS 读写两次，因此减少了 I/O 开销，提高了效率。

本文第 2 节介绍 kNN join 查询和已有的一些相关工作；第 3 节提出基于数据流的计算框架，并在该框架上实现 kNN join 算法；第 4 节给出了实验结果并分析。

2 问题定义和相关工作

2.1 kNN join 查询定义

给定 R^d 空间中两个数据集 R 和 S，R 和 S 分别表示一个 d 维的空间关系，其中每条记录 $r\in R(s\in S)$ 都是一个 d 维的点。R 为查询点集合，S 为被查询点集合。本文采用欧氏距离(euclidean distance)表示两个记录 r 和 s 之间的距离。

定义 1(kNN(r,S)) 给定一个查询对象 $r\in R$ 及正整数 k，从空间关系 S 中找出距离 r 最近的 k 个对象。

定义 2(kNNJ(R,S)) 给定两个空间关系 R 和 S

$knnJ(R,S)=\{(r,knn(r,S))\mid \text{for all } r\in R\}$

本文针对大规模空间数据集中两个空间数据集之间的 kNN join 算法进行研究，以减少算法的 I/O 开销，提高算法执行效率为优化目标。

2.2 相关工作

早期的 kNNJ 查询算法的研究都是在集中式的单节点单进程环境下进行的[12-14]。由于数据量的快速增长，单个节点已逐渐不能满足数据处理的要求，人们又相继提出了并行的 kNNJ 查询算法[15-18]。随着云计算的兴起，近年来不断有研究者提出了云环境下的 kNNJ 查询算法。

文献[8]提出了基于 Voronoi 图数据划分的 knnJ 查询的 MapReduce 算法。该算法通过仔细选择 Voronoi 图的支点(pivot)来将对象分配到不同的组，每个组在一个 Reduce 内执行 kNNJ 查询。尽管算法效率得到了提升，但算法受 pivot 点的选择方式和数量的影响较大。

Zhang 等首先提出了 Hadoop 平台下较高效的 kNN join 算法(H-zkNNJ)[9]。H-zkNNJ 改进和创新使用 Z-value 的方法，将多维空间的 kNN 查询转化为一维空间的范围查询，从而在 MapReduce 上较高效地实现了 kNN join 查询。H-zkNNJ 算法包括了 3 个具有串行执行的 MapReduce 任务：第 1 个 MapReduce 任务从 HDFS 中读取两张表 R 和 S 的数据，进行采样。构建 R 和 S 的随机移动 R_i 和 S_i，并确定 R_i 和 S_i 的划分值。第 2 个 MapReduce 任务将 R_i 和 S_i 划分到相应的块，并找出任一 $r\in R$，kNN(r,S)的候选点。第 3 个 MapReduce 任务从候选集中确定最终的 k 个点作为结果集。具体流程如图 1 所示。Job1、Job2、Job3 都对 HDFS 进行了读写，而 Job1、Job2 产生的输出仅是临时的中间结果。显然，该作业执行流程是低效的。

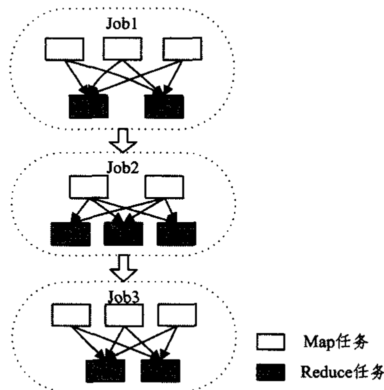

图 1 基于 MapReduce 的 kNN join 作业流程图

刘义等提出了 MapReduce 框架下基于 R-树的 kNN join 算法[10]。该算法首先需要构建分布式 R-树索引，然后利用 R-树索引进行 kNN join 查询，提高了查询效率。但构建分布式 R-树索引的代价较大，因此整个算法代价较高。

以上工作都是基于 MapReduce 计算框架实现的。MapReduce只是一种简单的数据处理模型，它将数据处理过程简化为 Map 和 Reduce 两个阶段，从而限制了 MapReduce 的计算表达能力。在 MapReduce 的设计中，Map 和 Reduce 任务的输出都需要写到磁盘上。尤其 Reduce 任务需要写到 HDFS 中，增大了 I/O 开销。文献[9,10]的算法都包括了 3 个串行的 Map-Reduce任务，每个 MapReduce 任务都需要读写 HDFS，当读写数据量很大时会产生大量不必要的 I/O 开销，降低了算法的效率。

3 基于数据流的框架下 kNN join 算法

3.1 基于数据流的计算框架及 IPO 运行模型

MapReduce 计算模型将计算过程抽象成 Map 和 Reduce 两个阶段，并通过混洗机制将两个阶段连接起来。但在一些应用场景中，为了套用 MapReduce 模型解决问题，不得不将问题分解成若干个有依赖关系的子问题，每个子问题对应一个 MapReduce 作业，最终所有作业形成一个有向图(Directed Acyclic Graph，DAG)。图 1 是 kNN join 作业的 DAG 图，包括了 3 个 MapReduce 任务，在该 DAG 中，由于每个节点是一个 MapReduce 作业，因此它们均会从 HDFS 上读一次数据和写一次数据(默认写 3 份)，使中间节点产生临时数据，进而会产生大量不必要的 I/O 浪费。

以上运行 DAG 作业的方式效率较低，根本原因是作业之间的数据不是直接流动的，而是借助 HDFS 作为共享数据

存储系统，即一个作业将处理后产生的数据写入 HDFS，另一个依赖于该作业的作业需再从 HDFS 上重新读取数据进行处理。显然更高效的方式是第一个作业直接将产生的数据传输给依赖它的作业，由此可以大大减少 I/O 使用率。

基于数据流的计算框架直接源于 MapReduce 框架，将 Map 和 Reduce 两个操作进一步拆分，即 Map 被拆分成 Input、Processor、Sort、Merge 和 Output，Reduce 被拆分成 Input、Shuffle、Sort、Merge、Processor 和 Output 等。这些分解后的元操作可以灵活组合，产生新的操作。这些操作经过组装，形成一个大的 DAG 作业。

基于数据流的计算框架不再按照 Map-Reduce 的运行模型，而是使用 Input-Processor-Output(IPO)的运行时模型。MapReduce 只是一种简单的数据处理模型，它将数据处理过程简化为 Map 和 Reduce 两个阶段，这限制了 MapReduce 的计算表达能力。基于数据流的计算框架不同，它可以包含任意多个数据处理阶段，提供了一种更加灵活高效的编程模型。IPO 运行时模型比 MapReduce 具有更好的可扩展性。

基于数据流的计算框架将数据处理不再按照单任务来建模，而是按照数据流图来处理。采用该计算框架之后则将作业 DAG 的依赖关系去除，将有依赖关系的作业转换为一个作业，如图 2 所示。

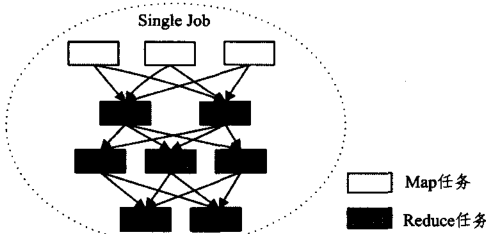

图 2 基于数据流框架的 kNN join 作业流程图

图 2 是对图 1 作业流程的优化，使用基于数据流的计算框架，对 MapReduce 作业依赖关系进行了裁剪，并将多个小作业合并成一个大作业，不仅计算量减少，而且对 HDFS 读写次数也会减少，从而提升了 DAG 作业的性能。

3.2 基于数据流的 kNN join 算法

本文提出基于数据流的 kNN join 算法，该改进算法利用空间填充曲线将多维数据映射为一维数据，从而将 k-近邻连接查询转化为一系列的一维范围查询。算法是在基于数据流的计算框架上实现的，称为 D-kNNJ (Data Stream based kNN join)。

算法 D-kNNJ 分为两个步骤：

步骤 1 基于采样技术和 z-order 曲线确定空间划分范围：通过取样寻找切割点以确定数据的划分范围，确保各个分块数据的均衡，同时确保在分块中满足查询条件。

步骤 2 基于数据流 kNNJ 查询：在分块中进行 kNN join 的查询。

在分布式环境中，为了提高复杂处理任务的执行效率及减少任务的运行时间，需要对输入数据集进行有效的划分。在数据划分过程中需要解决两个问题：1)数据偏斜，在分布式算法中，如果数据的划分不均匀，导致严重的数据偏斜，那么整个任务的执行时间将取决于执行任务最慢的节点所消耗的时间。数据的偏斜问题是影响算法效率的关键之一。2)在满足查询要求的同时减少数据的复制，由于 kNN join 查询中数据的紧耦合性，将可能满足查询要求的数据都划分到同一分块中，并且尽可能减少数据的复制是很困难的。D-kNNJ 算法在解决这两个问题时采用了传统的采样和空间填充曲线相结合的方法，在采样基础上，对样本点按照空间填充曲线 z-value 进行排序，根据等数据量原则，确定数据的划分。对于单个节点而言，其执行时间可能会受多重因素影响，例如处理器速度、内存大小等。本文仅关注所分配数据量的大小对节点任务执行时间的影响。

传统方法在处理数据量很大的 kNN join 时效率低下。本文采用 z-order 空间填充曲线的办法处理 kNN join，其基本思想是利用 z-order 空间填充曲线来进行任务的划分和 kNNJ 结果查询。

定理 1[19] 给定一个查询点 $q\in R^d$，数据集 $P\subset R^d$，以及常量 $\alpha\in R^d$。生成$(\alpha-1)$个随机变量$\{v_2,\cdots,v_\alpha\}$，$v_i\in R^d$，P 分别加上这些随机变量得到$\{P_1,\cdots,P_\alpha\}$，其中 $P_1=P$。算法 1 返回 kNN(q,P)的结果。

算法 1 zkNN(q,P,k,α)

输入：查询点 q，空间关系 P，常量 α，k

输出：q 在 P 中的 k 近邻 kNN(q,C(q))

1. $\{v_2,\cdots,v_\alpha\}$，$v_1=\vec{0}$，$v_i\in R^d$，$P_i=P+v_i(i\in[1,\alpha])$；
2. for i=1，…，α do
3. $q_i=q+v_i$，$C_i(q)=\emptyset$，z_{qi}为 q_i 的 z-value；
4. insert $z^-(z_{qi},k,p^i)$ into $C_i(q)$；
5. insert $z^+(z_{qi},k,p^i)$ into $C_i(q)$；
6. for each $p\in C_i(q)$，update $p=p-v_i$
7. $C(q)=\bigcup_{i=1}^{\alpha}C_i(q)=C_1(q)\cup\cdots\cup C_\alpha(q)$；
8. return kNN(q,C(q))

任何 $p\in R^d$，z_p 代表 p 的 z-value。Z_P 为集合 P 中所有点按 z-value 排好序的集合。其中 $z^-(z_{qi},k,p^i)$表示集合 Z_P 中 z_p 之前的 k 个点，$z^+(z_{qi},k,p^i)$表示集合 Z_P 中 z_p 之后的 k 个点。

算法 1 给出了利用 z-order 曲线进行 kNN 查询的过程。在分块中进行 kNN join 查询时，D-kNNJ 算法使用 B 树来加快查询的效率。

3.2.1 基于采样和 z-order 曲线的划分算法

在分布式环境下实现 kNN join 算法的关键在于将空间对象均匀划分到不同分区。基于采样和 z-order 曲线的划分算法的核心思想是：对数据集 R 和 S 进行采样，得到规模较小的数据集 R'和 S'。对数据集 R'和 S'中的点，按 z-value 进行排序。根据均匀划分的原则，确定分位点。样本 R'和 S'的分位点作为数据集 R 和 S 的分位点。

假设 R 的一个分区为 R_j，S 的一个分区为 S_j。当 S_j 中点的个数大于 $2k$ 时，对于 $r\in R_j$，可以在 S_j 找到 $C(r)$。当 S_j 中没有足够多的点时，对于 $r\in R_j$，查找 $C(r)$时需要在 S_{j-1}或者 S_{j+1}中去查找。为了确保在 S_j 中能够找到任一 $r\in R_j$ 的 $C(r)$，避免对多个分块查找，减少数据的复制和传输，需要将 S_{j-1}和 S_{j+1}中距离 S_j 最近的 k 个点分别复制到 S_j 中。

引理 1[8] 将 S_j 距离左右分位点最近的 k 个点复制到 S_j 中，则对于 $r\in R_j$，$C(r)$都包含在 S_j 中。

算法 2　基于采样和 z-order 曲线的划分算法

输入：数据集 R 和 S

输出：R 和 S 数据划分的范围

1. $p=1/(\varepsilon^2|R|)$, $R'=\emptyset$, $A=\emptyset$;
2. for each r∈R do
3. 　sample r into R' with probability p;
4. sort z-value of R' to be $Z_{R'}$;
5. for each $x\in Z_{R'}$ do
6. 　get x's rank s(x) in $Z_{R'}$;
7. 　x's rank r(x) in Z_R r(x)=s(x)/p;
8. for i=1,…,n−1 do
9. 　A[i]=x with closest r(x) value to (i/n) * |R|;
10. A[0]=0, A[n]=+∞;
11. for each s∈S do
12. 　sample s into S' with probiblity $p=1/(\varepsilon^2|S|)$;
13. $Z_{0,k-}=0$, $Z_{n,k+}=+\infty$;
14. for j=1,…,n−1 do
15. 　$Z_{j,k-}$=the (kp)th value to the left of A[j],
　　$Z_{j,k+}$=the (kp)th value to the right of A[j].

算法 2 对集合 R 按概率 p 进行采样，采样结果放入集合 R'(2−3 行)。对集合 S 按概率 p 进行采样，采样结果放入集合 S'(11−12 行)。将 R' 中点按 z-value 大小进行排序(4 行)。计算 R' 中点的 rank 值 $s(x)$，第 i 个点的 rank 值即为 i，并根据采样比率估算出 R' 中每个点在 R 中的 rank 值 $r(x)$(5−7 行)。选择 rank 值最接近 $(i/n)*|R|$ 的点作为分位点(8−9 行)。由于需要将 S_{j-1} 和 S_{j+1} 中距离 S_j 最近的 k 个点分别复制到 S_j 中，因此 S 的分位点与 R 的分位点并不相同，需要将 R 的分位点分别向左右移动 (kp) 个点(14−15 行)。

集合 R 和 S 的分位点信息将被保存在 HDFS 文件中，通过使用 Distributed Cache 共享给所有工作节点。

3.2.2　基于数据流的 kNNJ 查询算法

kNNJ 查询阶段包括了 3 个任务：Task1 对 R_i 和 S_i 进行数据划分，根据阶段 1 得到 R_i 和 S_i 的分位点信息对 R_i 和 S_i 进行数据的划分；Task2 在数据分块 S_{ij} 中查找对应分块 R_{ij} 中每个点 r 的 k 近邻候选点集 $C_i(r)$；Task3 根据 $C_i(r)$ 得出 R 中每个元素 r 的 kNN 集合 $C(r)$。

基于数据流的 kNNJ 查询算法流程如图 3 所示，对 DAG 的作业依赖关系进行了裁剪，并将 3 个 Task 合并成一个大作业(Single Job)，与基于 MapReduce 框架的 kNNJ 查询算法相比(见图 4)，不仅计算量减少，而且写 HDFS 次数也会减少，从而提升 DAG 作业的性能。MapReduce 的任务启动由 3 次减少为 1 次，大大降低了时间开销。

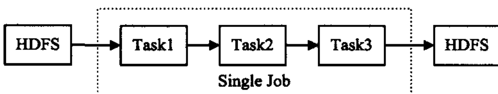

图 3　基于数据流框架的 kNNJ 查询算法流程图

图 4　基于 MapReduce 框架的 kNNJ 查询算法流程图

算法 3　基于数据流 kNNJ 查询算法

输入：数据分块 R_i 和 S_i

输出：R_i 中 r 与其在 S_i 中 k 近邻的键值对(r, kNN(r, C(r)))

1. for each $r\in R_i$ do
2. 　for j∈[1, n]
3. 　　if $A[j-1]\leqslant z_r\leqslant A[j]$
4. 　　insert r into block R_{ij}
5. for each $s\in S_i$ do
6. 　for j∈[1, n]
7. 　　if $z_{i,j,k-}\leqslant z_s\leqslant z_{i,j,k+}$
8. 　　insert s into block S_{ij}
9. for each r∈R do
10. 　for i=1,…,α do
11. 　　find $r\in R_{ij}$
12. 　　find $C_i(r)$ in S_{ij}
13. $C(r)=\bigcup_{i=1}^{\alpha}C_i(r)$
14. output (r, kNN(r, C(r)))

Task1 对应算法 3(1−8 行)将 R_i 和 S_i 按照阶段 1 得到 R_i 和 S_i 的分位点进行划分。Task2 和 Task3 对应算法 3(9−14 行)。Task2 使用两个向量分别存放 R_i 和 S_i 的分块 R_{ij} 和 S_{ij}，因为在混洗过程中进行了排序，所以向量中点都是按 z-value 排好序的。对于任一 $r\in R_{ij}$，可以使用二分查找法从 S_{ij} 中查找得到 $C_i(r)$。当数据量非常大、S_{ij} 规模超过内存大小时，为了加快查找效率，本文对 S_{ij} 建立了 B 树。Task3 对应算法 3(13−14 行)，从 Task2 的输出 $C_i(r)$ 中进行 kNN(r, $C(r)$)查找。在混洗阶段对 $C_i(r)$ 按照与 r 的距离大小进行排序，选出 Top k 个距离最小的点即为 kNN(r, $C(r)$)。

基于数据流 kNNJ 查询算法在 DAG 计算框架中的执行流程如图 5 所示。

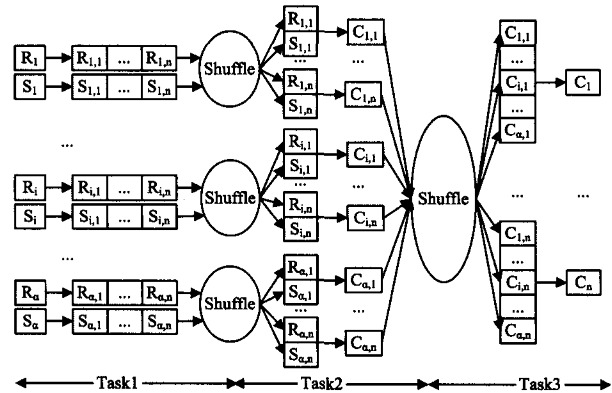

图 5　kNNJ 查询算法执行过程

3.3　代价分析

本文对 D-kNNJ 算法的通信、计算和 I/O 分别做了代价分析。

3.3.1　通信代价

阶段 1　基于采样技术和 z-order 曲线确定空间划分函数，从 R_i 和 S_i($i\in[1,\alpha]$)按照 $1/(\varepsilon^2|R|)$ 和 $1/(\varepsilon^2|S|)$ 概率进行采样，总通信代价为 $O(\alpha/\varepsilon^2)$。

阶段 2　kNNJ 查询，Task1 与 Task2 通信，将 R_i 和 S_i 划分到对应分块(R_{ij}, S_{ij})中，对于 $r\in R_i$，只会分到一个分块中，通信代价为 $O(\alpha|R|)$。对于 $s\in S_i$，有可能会被复制到两个分块中。在分块 S_{ij} 中，从邻近块中复制的点个数最多为 $2(k+\varepsilon|S|)$。通常 ε 取值很小，因此复制的数据量为 $O(k)$。对于 S_i，通信代价为 $O(\alpha(|S|+nk))$。Task1 与 Task2 总的通信

代价为 $O(\alpha(|R|+|S|+nk))$。Task2 与 Task3 通信，$r\in R_i$，r 的候选 k 近邻集 $C_i(r)$ 有 k 个点。所以总通信代价为 $O(\alpha k|R|)$。

D-kNNJ 算法两个阶段总的通信代价为 $O(\alpha(1/\varepsilon^2+|S|+|R|+k|R|+nk))$。$\alpha$ 为一个较小的常量，n、k、$1/\varepsilon^2$ 与 $|S|$、$|R|$ 相比都很小。因此 D-kNNJ 算法通信代价为 $O(1/\varepsilon^2+|S|+(k+1)|R|)$。

3.3.2 计算代价

阶段 1 需要对样本 R_i' 和 S_i' 按 z-value 进行排序，得到 R_i 划分点后计算 S_i 划分需要找到 $[kp]$ 个邻居节点，计算代价为 $O(n\log(|S_i'|))$。因此阶段 1 总的计算代价为 $O(\alpha(|R|+|S|+|R'|\log|R'|+|S'|\log|S'|+n\log(|S'|)))$，即 $O(\alpha(|R|+|S|+(1/\varepsilon^2)\log(1/\varepsilon^2)+n\log(1/\varepsilon^2)))$。

阶段 2 Task1 计算代价忽略不计。Task2 对 S_{ij} 进行二分查找时计算代价为 $O((|R_{ij}|+|S_{ij}|)\log|S_{ij}|)$。如果阶段 1 的划分能够使得负载均衡，则计算代价为 $O((|R|/n+|S|/n)\log(|S|/n))$，Task2 总的计算代价为 $O(\alpha(|R|+|S|)\log(|S|/n))$（如果负载不均衡，最坏的情况下，总计算代价为 $O(\alpha(|R|+|S|)\log|S|)$）。Task3 的时间复杂度为 $O(\alpha k|R|\log k)$。

D-kNNJ 算法两个阶段总的计算代价为 $O(\alpha((1/\varepsilon^2)\log(1/\varepsilon^2)+n\log(1/\varepsilon^2)+(|R|+|S|)\log|S|+k|R|\log k))$，由于 α 为数值很小的常量，并且 $k|R|\log k$ 远小于 $(|R|+|S|)\log|S|$，因此总计算代价即为 $O(\alpha((1/\varepsilon^2)\log(1/\varepsilon^2)+n\log(1/\varepsilon^2)+(|R|+|S|)\log|S|))$。

3.3.3 I/O 代价

阶段 1 需要从 HDFS 读取 R 和 S，I/O 代价为 $O(|R|+|S|)$。向 HDFS 写入分位点信息，I/O 代价为 $O(3\alpha n)$（因为 HDFS 需要写 3 份）。

阶段 2 需要从 HDFS 读取 R 和 S，I/O 代价为 $O(|R|+|S|)$。向 HDFS 写入最终结果 $(r,\mathrm{kNN}(r,C(r)))(r\in R)$，I/O 代价为 $O(3k|R|)$。

D-kNNJ 算法两个阶段总的输入代价为 $O(2(|R|+|S|))$，输出代价为 $O(3\alpha n+3k|R|)$。由于 α、n 均为较小的常数，因此输出代价为 $O(3k|R|)$。

在数据量特别大的情况下，I/O 已成为性能瓶颈，H-zkNNJ 输入代价为 $O((\alpha+1)(|R|+|S|)+\alpha k|R|)$，输出代价为 $O(3((\alpha+\alpha k+k)|R|+\alpha|S|))$。与 H-zkNNJ 算法相比，D-kNNJ 算法的 I/O 代价明显减小，提高了整个算法的执行效率。

4 实验结果与分析

4.1 实验设置

集群有 8 个节点，CPU：Xeon E5-2620（双核 2.00GHz），内存：8G，硬盘：6T（硬盘实际可用空间为 44.03T）；操作系统：CentOS6.4(64bit)；分布式文件系统：HDFS；DAG 框架是基于 MapReduce 框架之上的，其中一个节点为 master，剩下节点均为 worker 节点。

本文数据采用来自 OpenMapStreet 项目的真实数据集[8,10]。每个数据集表示美国一个州的道路网信息。整个数据集包括了美国 50 个州的信息，超过 160000000 条记录，数据量达到 6.6GB。每条记录包括：记录标识 ID、二维坐标和其他描述信息。为了测试性能并与其他算法进行性能对比，从数据集中随机提取两个样本集 R 和 S，其中，R 和 S 的数据规模分别为 10M、20M、40M 和 80M。采用 $M\times N$ 标识数据配置，其中 M 和 N 分别表示 R 和 S 中的数据条数（百万为单位），如 (40×80) 表示 R 中有 40M 条记录，S 中有 80M 条记录。默认采样概率 $\varepsilon=0.003$。

4.2 结果分析

(1)数据量对查询效率的影响

图 6 示出在不同 k 值条件下，算法的执行时间。从图中可以看出，在不同的 k 值条件下，随着数据规模 $|R|*|S|$ 的不断增大，算法查询时间随数据规模增长呈近线性的增长。当 $k=10$ 时，数据量为 40M * 40M 时，算法执行时间为 540.25s。数据量为 80M * 80M 时，算法执行时间为 1017.30s。计算任务增长了 4 倍，但其执行时间仅增长了 1.8 倍。当 k 为其他值时，算法表现出类似的结果，即随着数据量的增长，算法的执行时间没有出现快速增长。因此，算法表现出了非常好的可扩展性能。

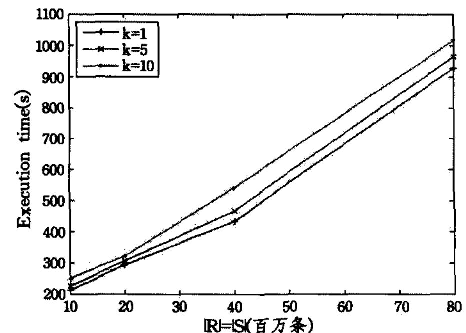

图 6 不同 k 值下时间性能随数据规模变化

(2)k 对查询效率的影响

图 7 示出了在不同数据规模条件下，算法的执行时间。从图中可以看出，在不同数据规模下，随 k 值的增长，算法执行时间呈近线性的增长。由于 k 值增大，导致了查询的中间结果和最终结果集都会增大，因此阶段 2 中 Task2、Task3 都会产生更大的 I/O 和 CPU 代价，整个算法的执行时间都会增加。

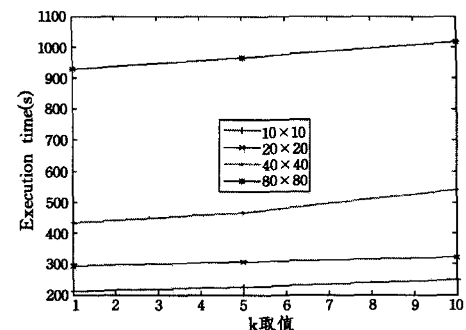

图 7 不同数据规模下时间性能随 k 值变化

(3)节点数目对查询效率的影响

图 8 示出了在不同数据规模条件、不同节点数目 N 值下，算法的执行时间。从图中可以看出，在不同数据规模下，随 N 值的增长，算法执行时间呈近线性的递减，表现出良好的可扩展性。计算节点是衡量可扩展性的一个重要维度，若计算任务不变，当计算节点增加一倍，则执行时间应当减少一

半。然而在实际算法运行过程中，受容错、数据通信等各方面因素的影响，其执行时间以一种近线性的速率递减。当数据量较小时，如图中数据量为 10M * 10M 和 20M * 20M 时，随着节点的增加，会出现资源闲置的情况，效率并不能得到进一步的提升。

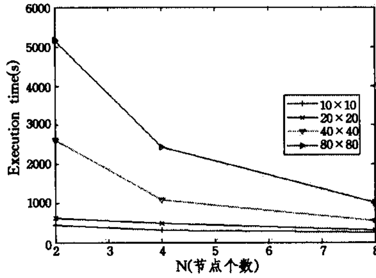

图 8　$k=10$ 时不同数据规模下时间性能随 N 值变化

(4)与 H-zkNNJ 的性能比较

H-zkNNJ 算法是 MapReduce 框架下面向大规模数据的 kNNJ 查询算法。由于其是基于 MapReduce 框架的，每个 MapReduce 任务都需要读写 HDFS，即使某些 MapReduce 任务产生的数据仅是临时数据。显然，这种表达作业依赖关系的方式的效率不高。

在进行对比时，固定 k 值为 10，数据规模从 10M * 10M 变化到 80M * 80M。图 9 示出了固定 $k=10$ 时，在不同数据规模条件下，D-kNNJ 算法与 H-zkNNJ 算法性能的对比实验结果，显然 D-kNNJ 的性能要优于 H-zkNNJ。随着数据量的增大，基于数据流的计算框架比基于 MapReduce 框架的优势将会更加明显，这是由于基于数据流的计算框架减少了任务间数据的读写，从而大大降低了 I/O 开销。

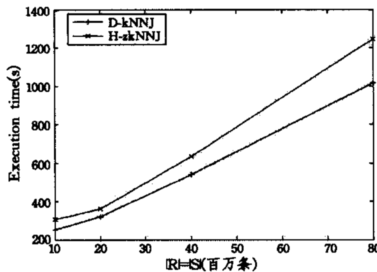

图 9　$k=10$ 时与 H-zkNNJ 算法性能比较

图 10 示出了在数据规模固定为 80M * 80M，k 值分别取 1，5，10 时，D-kNNJ 算法与 H-zkNNJ 算法性能的对比实验结果。无论 k 取何值时，D-kNNJ 的性能要优于 H-zkNNJ。且随着 k 值的增大，中间结果数据量也会随之变大。因此 H-zkNNJ 算法对中间结果的 I/O 开销变大，这就导致了 H-zkNNJ 算法效率低于 D-kNNJ 算法。

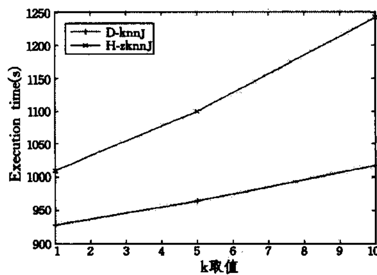

图 10　不同 k 值时与 H-zkNNJ 算法性能比较

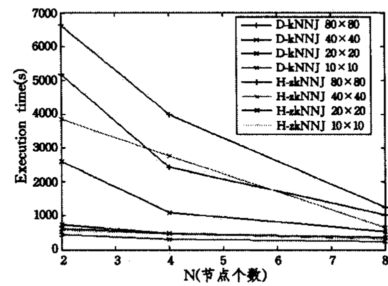

图 11　$k=10$ 时不同节点数目和数据规模下与 H-zkNNJ 算法性能比较

图 11 示出了在节点数目分别为 2、4、8 时，不同数据规模下，D-kNNJ 算法与 H-zkNNJ 算法的执行时间。图中结果表明，本文提出的 D-kNNJ 算法在不同节点数目、不同数据规模的情况下，效率均要优于 H-zkNNJ 算法。随着节点数目的增多，不同数据规模的执行时间均得到减少，数据规模越大执行时间减少得越明显。对于数据规模较小的情况，随着节点数目增多，出现了资源过剩、效率提高不再明显的现象。在数据量规模较大时，产生的中间结果也较大，因此中间结果的存取所产生的 I/O 开销也较大，此时，D-kNNJ 算法比 H-zkNNJ 算法的优势体现得比较明显。但数据量较小、中间结果集也不大时，D-kNNJ 算法的优势便不再明显了。

结束语　本文研究分布式环境下大规模空间数据集的 kNN join 查询问题，MapReduce 框架下由于每个 MapReduce 任务都需要读写分布式文件系统，因此 MapReduce 表达作业之间依赖关系时是低效的，这就导致现有基于 MapReduce 的算法效率不高。本文首先在 MapReduce 框架的基础上提出了基于数据流的框架，并在该框架上提出了一种 kNN join 查询算法 D-kNNJ，该算法将 kNN join 查询归结为两个步骤，从而可以减少读写分布式文件系统的次数，有效降低了查询过程中的 I/O 代价。D-kNNJ 能高效处理大规模空间数据的 kNN join 查询问题，具有良好的可扩展性和查询效率。实验表明 D-kNNJ 算法能很好地支持大规模空间数据的 kNN join 查询处理，具有实用价值。在性能对比上，D-kNNJ 算法也优于 H-zkNNJ 算法，效率提高了近 30%。下一步工作将研究多轮的 kNN join 查询算法的性能优化。

参考文献

[1] Böhm C, Krebs F. The k-nearest neighbour join: Turbo charging the KDD process[J]. Knowledge and Information Systems, 2004, 6(6): 728-749

[2] Kavraki L E, Plaku E. Distributed Computation of the knn Graph for Large High-Dimensional Point Sets[J]. Parallel Distributed Computation, 2007, 67(3): 346-359

[3] Sardana D, Bhatnagar R. Graph Clustering Using Mutual K-Nearest Neighbors[M]// Active Media Technology. Springer International Publishing, 2014: 35-48

[4] Xia C, Lu H, Ooi B C, et al. Gorder: an efficient method for KNN join processing[C]// Proceedings of the Thirtieth international conference on VLDB Endowment. 2004: 756-767

[5] Yu C, Cui B, Wang S, et al. Efficient index-based KNN join processing for high-dimensional data[J]. Information and Software Technology, 2007, 49(4): 332-344

[6] Cheung K L, Fu A W C. Enhanced nearest neighbour search on

the R-tree[J]. ACM SIGMOD Record,1998,27(3):16-21

[7] Dean J,Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[8] Zhang C,Li F,Jestes J. Efficient parallel kNN joins for large data in MapReduce[C]// Proceedings of the 15th International Conference on Extending Database Technology. ACM,2012:38-49

[9] Lu W,Shen Y,Chen S,et al. Efficient processing of k nearest neighbor joins using mapreduce[J]. Proceedings of the VLDB Endowment,2012,5(10):1016-1027

[10] 刘义,景宁,陈荦,等. MapReduce 框架下基于 R-树的 k-近邻连接算法[J]. 软件学报,2013,24(8):1836-1851

[11] Apache. Hadoop[EB/OL]. 2014-4-10[2014-4-22]. http://hadoop. apache. org/

[12] Liu Y,Cui J,Huang Z,et al. SK-LSH: An Efficient Index Structure for Approximate Nearest Neighbor Search[J]. Proceedings of the VLDB Endowment,2014,7(9):745-756

[13] Choi W,Oh S. Fast nearest neighbor search algorithm using the cache technique[J]. Advanced Robotics, 2013, 27(15): 1175-1187

[14] Gieseke F,Heinermann J,Oancea C,et al. Buffer kd trees: processing massive nearest neighbor queries on GPUs[C]// Proceedings of The 31st International Conference on Machine Learning. 2014:172-180

[15] Luo G,Naughton J F,Ellmann C J. A non-blocking parallel spatial join algorithm[C]// Proceedings of the 2002 IEEE 18th International Conference on Data Engineering. 2002:697-705

[16] 何洪辉,王丽珍,周丽华. pgi-distance:一种高效的并行 KNN-join 处理方法[J]. 计算机研究与发展,2007,44(10):1774-1781

[17] Mutenda L, Kitsuregawa M. Parallel r-tree spatial join for a shared-nothing architecture[C]// Proceedings of the 1999 IEEE International Symposium on Database Applications in Non-Traditional Environments. 1999:423-430

[18] Yu C,Cui B,Wang S,et al. Efficient index-based KNN join processing for high-dimensional data[J]. Information and Software Technology,2007,49(4):332-344

[19] Yao B,Li F,Kumar P. K nearest neighbor queries andknn-joins in large relational databases (almost) for free[C]// Proceedings of the 2010 IEEE 26th International Conference on Data Engineering. 2010:4-15

(上接第 172 页)

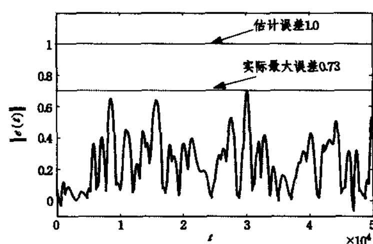

图 3 $\|e(t)\|$ 的波动范围

结束语 目前,混沌同步的研究中,通常假设同步的两个系统是一样的,没有任何的参数失配。由于工艺精度、环境温度变化等因素的影响,完全相同的混沌系统是不存在的,参数失配是不可避免的,且对同步的质量有决定性的影响。因此研究和设计参数失配条件下依然可以鲁棒同步的控制系统对于混沌保密通信系统走向实际应用具有非常重要的意义。参数失配时,同步变为拟同步。本文主要研究了小范围参数失配下的脉冲同步问题,但是在多大的参数失配范围内系统仍然可以实现拟同步还需要更为具体的研究。

参考文献

[1] 柴秀丽,王玉璟,袁光耀,等. 未知干扰下混沌系统的修正函数投影滞后同步[J]. 计算机科学,2014,41(4):283-301

[2] 方洁,邓玮,姜长生,等. 具有扇区非线性输入的混沌系统函数投影同步[J]. 系统工程与电子技术,2012,34(9):1872-1877

[3] 杨叶红,肖剑,马珍珍. 部分线性的分数阶混沌系统修正函数投影同步[J]. 物理学报,2013,62(18):180505-1-7

[4] Sudheer K,Sebastian Sabir M. Adaptive modified function projective synchronization of multiple time-delayed chaotic Rossler system[J]. Physics Letters A,2011,375(8):1176-1178

[5] Liu C,Li C D,Li C J. Quasi-synchronization of delayed chaotic systems with parameters mismatch and stochastic perturbation [J]. Communications in Nonlinear Science and Numerical Simulation,2011,16:4108-4119

[6] Han Q,Li C D,Huang J J. Anticipating synchronization of chaotic systems with time delay and parameter mismatch[J]. Chaos,2009,19:013104-1-10

[7] Han Q,Li C D,Huang J J. Estimation on error bound of lag synchronization of chaotic systems with time delay and parameter mismatch[J]. Journal of Vibration and Control,2010,16(11): 1701-1711

[8] He W L,Qian F,Han Q L,et al. Lag quasi-synchronization of coupled delayed systems with parameter mismatch[J]. IEEE Transactions on Circuits and Systems-I: Regular papers,2011, 58(6):1345-1357

[9] He W L, Qian F, Han Q L. Delay-dependent synchronization criteria for delayed chaotic systems with parameter mismatches[C]// Proceedings of the 30th Chinese Conference. Yantai, China, 2011:22-24

[10] He W L,Qian F,Han Q L,et al. Synchronization error estimation and controller design for delayed Lur'e systems with parameter mismatches[J]. IEEE Transactions on Circuits and Systems I: regular papers,2012,23(10):1551-1563

[11] Li C D, Liao X F, Yang X F, et al. Impulsive stabilization and synchronization of a class of chaotic delay systems[J]. Chaos, 2005,15(4):043103-1-9

[12] He W L, Qian F, Cao J D, et al. Impulsive synchronization of two nonidentical chaotic systems with time-varying delay[J]. Physics Letters A,2011,375:498-504

[13] Zhang H G,Ma T D,Huang G B. Robust global exponential synchronization of uncertain chaotic delayed neural networks via dual-stage impulsive control[J]. IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics,2010,40(3):831-844

[14] Cao J D,Ho Daniel W C,Yang Y Q. Projective synchronization of a class of delayed chaotic systems via impulsive control[J]. Physics Letters A,2009,373(35):3128-3133