MapReduce 框架下的优化高维索引与 KNN 查询

梁俊杰[1],李凤华[2,3],刘琼妮[1],尹 利[1]
(1. 湖北大学计算机与信息工程学院,湖北武汉 430062;2. 中国科学院信息工程研究所信息安全国家重点实验室,北京 100093;
3. 北京电子科技学院,北京 100070)

摘 要: 针对大规模高维数据近似查询效率低下的问题,利用 MapReduce 编程模型在大规模集群上的数据与任务的并行计算与处理优势,提出 MapReduce 框架下大规模高维数据索引及 KNN 查询方法(iPBM),重点突破 MapReduce 数据块(block)的优化划分与各数据块对计算的共同贡献两大难题,利用两阶段数据划分策略并依据相关性与并行性原则将数据均匀分配到各数据块中,设计分布式的双层空间索引结构与并行 KNN 查询算法,检索时利用全局索引、局部索引与二维位码索引实现三层数据过滤,大幅缩小搜索范围并降低高维向量计算代价,实验表明 iPBM 对大规模高维数据的近似查询具有准确性、高效性和扩展性.

关键词: 云计算; MapReduce; KNN 查询; 高维索引
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2016)08-1873-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.08.015

Optimized High-Dimensional Index and KNN Query in MapReduce

LIANG Jun-jie[1], LI Feng-hua[2,3], LIU Qiong-ni[1], YIN Li[1]
(1. *Department of Computer Science and Technology, Hubei University, Wuhan, Hubei* 430062, *China*;
2. *State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093, *China*;
3. *Beijing Electronic Science&Technology Institute, Beijing* 100070, *China*)

Abstract: To address the low efficiency problem caused by the approximate large-scale high-dimensional data query, we propose a novel high-dimensional index and KNN query method, called iPBM, which exploits two main problems, including the optimal division on the MapReduce's data block and their contributions to the computing. Specifically, based on the principles of relativity and parallelity, iPBM employs a two-phase partitioning scheme of clustering and zoning to equally split the data to the available blocks, then we design a distributed two-layer index structure and parallel KNN query algorithm. With fully considering the global index, local index and two-dimensional bitcode property, iPBM achieves triple-layers filtering, and thus the number of queried area and the computing cost on the high-dimensional data is minimized. The accuracy, efficiency and scalability of the proposed iPBM are thoroughly evaluated via detailed simulations.

Key words: cloud; MapReduce; KNN query; high-dimensional index

1 引言

随着下一代互联网、物联网和云计算[1]等信息技术的创新发展和重点应用,用户面临海量数据或大数据诸多挑战,很多领域数据呈现高维表现形态,如金融交易数据、多媒体数据、航天数据、生物数据等,维度高达几十维甚至上百维. 高维空间数据应用常见以近似查询(similarity query)为主,如何提高大规模高维数据近似查询效率已成为学术界的研究热点.

为适应大数据使用管理需求,Google 公司提出了 MapReduce 计算模型[2,3],将运行于大规模集群上的复杂并行计算过程高度地抽象为两个函数:Map 和 Reduce. 一个 Map/Reduce 作业(Job)通常会把输入的数据集切分为若干独立的数据块,由 Map 任务(Task)以完全并行的方式处理它们,结果输入给 Reduce 任务. MapReduce 框架由一个单独的 Master 和集群节点上的

收稿日期:2014-12-31;修回日期:2015-11-08;责任编辑:诸叶梅
基金项目:国家发改委 2012 年信息安全专项(No. 发改办高技[2013]1309);国家自然科学基金(No. 61170251);湖北省自然科学基金重点项目(No. 2013CFA115);武汉市科技攻关计划(No. 2013012401010851)

Slave 共同组成. Master 负责调度构成一个作业的所有任务,这些任务分布在不同的 slave 上. 这种框架的设计可以充分利用整个集群的网络带宽,同时保障了系统可靠性和容错性,是目前云计算环境的核心计算模式.

高效的数据存取策略是提高高维数据查询性能的关键,为了避免不必要的数据存取普遍做法是利用索引技术. 在大数据时代,如何将高维索引技术和并行计算模型有机结合是一个新的研究方向,最近几年针对云计算环境下的索引技术研究受到学者的广泛关注[4~8]. Dittrich 等人分别在 2010 年和 2012 年 VLDB 会议上提出了Hadoop ++[9]和 HAIL[10]. Hadoop ++ 系统使用了一种称为 Trojan 的索引结构,将描述输入片段(input splits)的索引信息写入片段中,在一个片段中可以有多个局部索引(partial index),每个局部索引只描述片段中部分数据信息,在这些局部索引中有一个主索引(primary index),查询时根据需要查找相应的索引. Trojan 索引的创建过程是在数据加载阶段完成,因此不影响查询效率. HAIL 是为缩短Hadoop ++ 索引构造时间,对每个 Hadoop 备份数据创建了一个聚类索引,检索时选择最佳的索引而提高查询效率. Hadoop ++ 和 HAIL 都是在 Hadoop 默认的数据块划分基础上建立索引,需要进一步考虑查询需求和数据分布对数据划分的影响. Eltabakh 等人对 Hadoop 进行扩展提出了 CoHadoop[11],定义文件级属性(file-level property)概念称为 locator,在 NameNode 上新增一个 locator 数据表,将 locator 属性值相同的文件存储到同一个 DataNode 上,达到将相关数据存储在同一节点上的目的,但是如何实现系统自动判断两个文件是否相关是 CoHadoop 的难点. Zaschke 发表在 2014 年 SIGMOD 会议上的论文,描述了一种多维数据存储和索引结构称为 PH 树[12],为减少存储空间在索引结构中共享二进制描述前缀,PH 树支持点查询和范围查询时快速数据存取,但 PH 树如何扩展到 MapReduce 框架有待进一步研究. 文献[13 ~ 15]是在 MapReduce 框架的计算节点上建立局部 B + -Tree、R-Tree 或四叉树索引,并基于局部索引构造全局索引,查询时依据查询模型动态选择合适的索引,这些方法都是基于 MapReduce 默认的数据划分无法保障数据块对计算共同贡献. Wei 等人提出了基于 Voronoi 图数据划分的 knnJ 查询的 MapReduce 算法,将不同 Voronoi 图的对象分配到不同的组内,但是算法效率对 Voronoi 图支点选择策略具有很强的依赖性[16]. Doulkeridis 等探讨了利用 MapReduce 处理 Top-K 查询的一些基本问题,但是缺少对问题的深入分析和实验验证[17]. 国内,刘义等人在2013 年发表论文[18,19],提出了一种采样算法以快速确定空间划分函数,构建了符合 MapReduce 计算模型的索引结构,并利用分布式 R-树索引实现 K-近邻连接查询,该方法主要限定于地理空间的 knnJ 查询,没有测试高维空间对算法性能的影响.

针对目前鲜有人研究 MapReduce 框架下的高维数据索引及 KNN 检索方法[20],MapReduce 框架自身缺少数据分块划分优化策略以及无法保证每个数据分块对计算共同贡献的缺陷,本文提出了一种新的 MapReduce 框架下的高维索引(Index Partition Bitcodes in MapReduce,简称 iPBM). 充分利用 MapReduce 框架在大规模集群上的数据和任务的并行计算与处理能力,基于高维数据分布和近似查询特点优化高维空间数据块的划分机制,设计可扩展的、分布式的双层空间索引以及 KNN 并行查询算法,检索过程实现了数据快速筛选避免了大量数据无效计算,实验表明 iPBM 在提高高维数据查询效率和扩展性方面都具有明显优势.

2 问题描述和背景知识

为了简化问题阐述,本文引用的符号和含义如表 1 所示,并作如下合理的假设:

(1)针对分析型应用环境,数据集相对稳定. 因此,索引只需一次构建,用于多次查询.

(2)MapReduce 始终保持负载均衡,当数据量发生变化时,MapReduce 可以适当增加或减少服务器数量或者调整配置参数.

(3)高维向量相似性用相应的空间点距离来表示. 不失一般性,本文采用欧氏(Euclidean)距离.

(4)文中以 KNN 查询为例介绍高维数据近似查询实现过程,其他种类的近似查询可以依此类推,限于篇幅不做详述.

表 1 相关符号描述

符号	具体含义
$DS=\{\boldsymbol{P}_1,\boldsymbol{P}_2,\cdots,\boldsymbol{P}_n\}$	高维向量数据集
d	数据维度
$\boldsymbol{P}(p_1,p_2,\cdots,p_d)$	点对象 P 的高维向量表示
$P(D_p,B_p)$	点对象 $\boldsymbol{P}$ 的二维向量表示,即位码索引值,D_P 表示 $\boldsymbol{P}$ 与参考点的距离,B_P 表示位码值
$C=\{C_1,C_2,\cdots,C_m\}$	数据簇(cluster)集合
$O=\{\boldsymbol{O}_1,\boldsymbol{O}_2,\cdots,\boldsymbol{O}_m\}$	数据簇的质心集合
$AS=\{A_1,A_2,\cdots,A_{2d}\}$	数据簇中划分的分区(Area)集合
$A(b_1,b_2,\cdots,b_d)$	分区的位码表示

定义 1 KNN 查询 假设高维向量对象集 $DS=\{\boldsymbol{P}_1,\boldsymbol{P}_2,\cdots,\boldsymbol{P}_n\}$,查询点 $\boldsymbol{Q}$,则 KNN 查询返回 DS 中与 $\boldsymbol{Q}$ 距离最近的 k 个对象,并按距离升序排列:$\mathrm{KNN}(DS,\boldsymbol{Q},k)=\{\boldsymbol{P}_r,\boldsymbol{P}_t,\cdots,\boldsymbol{P}_s\mid\boldsymbol{P}_r,\boldsymbol{P}_t,\boldsymbol{P}_s\in DS\}$.

3 MapReduce 框架下的高维索引

iPBM 是在 MapReduce 的开源框架 Hadoop 平台下

实现的,KNN 查询分为两个阶段:

(1)索引构建阶段:对海量高维数据进行聚类和分区划分,针对 Master 和 Slave 分别设计不同的索引结构. (2)KNN 查询阶段:利用双层索引确定候选点对象集和任务划分,执行并行 KNN 查询.

3.1 数据划分

MapReduce 框架由一个提供元数据服务的 Master 节点和若干个提供存储的 Slave 节点组成,适合大规模数据并行计算和任务处理. 为了适应这种机制,创建索引之前有必要对数据集进行必要的预处理,使得数据应尽可能均匀地划分到多个数据块中,并对最终结果均有贡献.

3.1.1 聚类划分

数据划分首先是对数据集 DS 进行聚类分析. 不失一般性,我们采用 K – means 聚类方法[21]对 d 维空间数据进行全局分析,得到数据簇 $C=\{C_1,C_2,\cdots,C_m\}$ 及质心 $O=\{\boldsymbol{O}_1,\boldsymbol{O}_2,\cdots,\boldsymbol{O}_m\}$.

3.1.2 分区划分

数据空间聚类划分后,对各个数据簇按其数据分布特征做进一步的分区划分,得到数据簇的分区 $AS=\{A_1,A_2,\cdots,A_{2^{d'}}\}$. 为方便描述,本文将数据簇质心作为簇的参考点用于计算分区和点对象的位码索引值,也可以选择其他点作为参考点.

定理 1　划分分区的维数　假设数据簇的大小 $Size(C)$、点对象的大小 $Size(\boldsymbol{P})$ 以及查询结果集 K 值,则划分数据簇分区的维数:

$$d'=\left|\log_2\frac{Size(C)}{Size(\boldsymbol{P})\times K\times \mathrm{c}}\right|,\ d'\leqslant d$$

公式中 c 是常量($\mathrm{c}\geqslant 1$),它的作用是使划分后的分区内包含的点对象数量至少是 K 的 c 倍,这样每个候选簇只需确定一个候选分区就能基本满足查询结果对象个数要求,避免了搜索多个分区的计算复杂性. 以我们的实验数据二十维向量空间为例,数据集 *DS* 大小 540MB,点对象大小为 104Byte. 根据 K-means 聚类算法 *DS* 被划分成 4 个数据簇 A、B、C、D,其中数据簇 B 大小为 250MB. 假设 K = 10,c = 20,根据定理 1 计算用于划分数据簇 B 分区的维数为 13,簇 B 被划分为 2^{13} 个分区.

定义 2　分区位码 $A(b_1,b_2,\cdots,b_{d'})$　假设整数变量 i 代表维标($1\leqslant i\leqslant d'$),则 b_i 表示分区 A 的第 i 维位码、o_i 表示质心 $\boldsymbol{O}$ 的第 i 维位码、p_i 表示任意点 $\boldsymbol{P}$ 的第 i 维位码,假设 A 所在的数据簇质心为 $\boldsymbol{O}(o_1,o_2,\cdots,o_{d'})$,由于同一分区内所有点对象具有相同的位码值,因此可以由区内任意一点 $\boldsymbol{P}(p_1,p_2,\cdots,p_{d'})$ 的位码值得到分区 A 的位码:

$$b_i=\begin{cases}1, p_i\geqslant o_i\\ 0, \text{otherwise}\end{cases},\ 1\leqslant i\leqslant d'$$

由定义 2 可以看出,一个分区的位码表达该分区相对簇参考点的位置关系,在数据簇内任一分区均可用长度为 d' 的唯一位码字符串来表示. 以二维向量空间为例,各个分区的位码表示如图 1 所示.

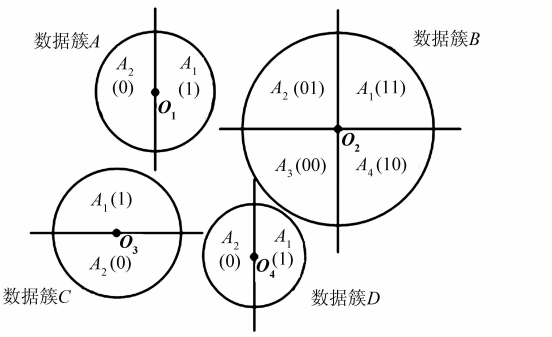

图1 数据划分示意图

为有效克服高维空间“维数灾难[22]”影响,iPBM 综合利用近似向量表示法和一维向量转换法的优点,利用二维的位码索引值压缩高维向量表示.

定义 3　位码索引值　高维向量点对象 $\boldsymbol{P}(p_1,p_2,\cdots,p_d)$ 的位码索引值为 $\boldsymbol{P}(D_p,B_p)$,其中 D_p 是 $\boldsymbol{P}$ 与所属簇参考点间的距离;B_p 是 $\boldsymbol{P}$ 与参考点间的近似位置关系,即 $\boldsymbol{P}$ 的位码值.

由定义 3 可知,位码索引值的主要作用体现在检索过程中不仅可以根据 B_p 过滤候选分区,而且可以根据 D_p 进一步过滤候选分区环,从而在没有任何高维欧氏距离计算情况下就能实现搜索范围两级过滤,大大缩小搜索范围从而降低高维向量距离计算代价.

3.2 数据存储

在数据存储时为了兼顾相关性和并行性两方面原则,iPBM 将一个簇数据的所有 Block 放在一个 Slave 节点上,一个 Slave 节点可以存储多个簇的数据,同时将簇中所有分区的数据均匀分配到该簇的每个 Block 中. 这样做的好处是,查询时筛选出候选分区后,根据候选分区的对象信息存储在多个 Block 中的特点,可以利用多个 Map 任务并行计算,提高查询并行性和查询效率. 在本文第四章 KNN 查询算法介绍中,可以看出这种数据划分和存储的好处.

在 iPBM 中,在 Master 节点上设计了一个定位表存储各个数据簇在 Slave 节点上的分布信息. 如图 2 和图 3 所示,定位表指示簇 A 和簇 D 存储在 Slave1 上,簇 B 存储在 Slave2 上,簇 C 存储在 Slave3 上,则簇 A 的数据块 A1 和 A2 以及簇 D 的数据块 D1 存放在 Slave1 上,簇 B 的数据块 B1、B2 和 B3 存放在 Slave2 上,簇 C 的数据块 C1 和 C2 存放在 Slave3 上. 另外的 3 个 Slave 节点存储这些簇的备份.

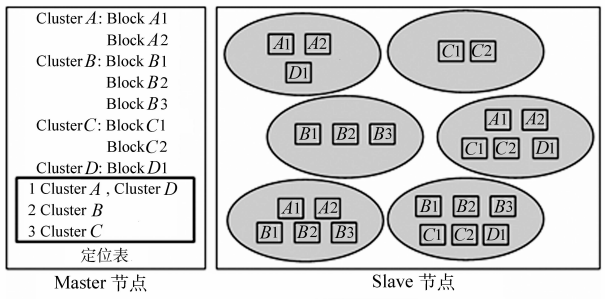

图2 逻辑存储结构图

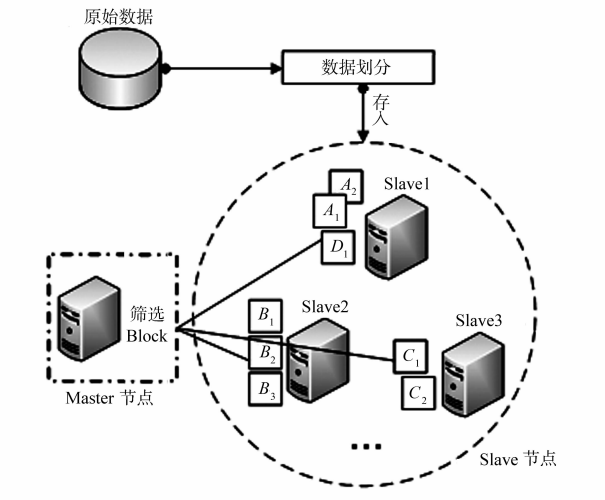

图3 物理存储结构图

3.3 索引结构

依据 iPBM 的数据划分和存储结构,为 KNN 查询设计基于高维空间位置编码的双层索引 $G-L$(图 4),包含全局索引 $G-Index$ 和局部索引 $L-Index$. 全局索引存放在 Master 节点上负责数据定位. 局部索引存放在 Slave 节点上负责数据存取.

定义 4 全局索引 $G-Index$ 以键值对形式存储数据簇元数据信息,表示为 $<\boldsymbol{O}_i(o_{i1},o_{i2},\cdots,o_{id}),R_i>$. 其中 $\boldsymbol{O}_i(o_{i1},o_{i2},\cdots,o_{id})$ $(i=1,2,\cdots,m)$ 为第 i 个数据簇的质心坐标,R_i 为该簇的覆盖半径.

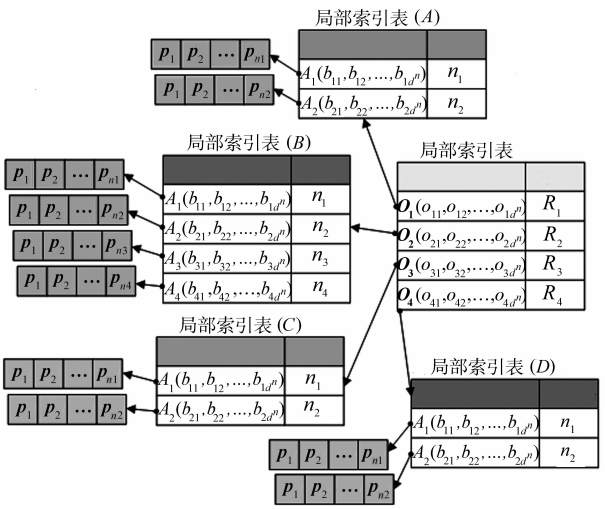

图4 G-L 双层索引结构

定义 5 局部索引 $L-Index$ 以键值对形式存储数据簇内的分区和点对象信息,表示为 $<A_j(b_{j1},b_{j2},\cdots,b_{jd'}),n_j>$. $A_j(b_{j1},b_{j2},\cdots,b_{jd'})(j=1,2,\cdots,2^{d'})$ 为数据簇的第 j 个分区的位码,$n_j(n_j>0)$ 为该分区内的点对象个数.

以图 1 数据划分示意图,双层索引 $G-L$ 结构如图 4 所示.

4 基于 G-L 索引的 KNN 查询

4.1 数据筛选

定义 6 候选分区环 假设查询点 Q 在候选簇 $C<O(o_1,o_2,\cdots,o_d),R>$ 的位码索引值为 $Q(D_Q,B_Q)$,则候选分区与查询范围(Q,r)相交的局部环状区域为候选分区环,候选分区环内的任一点对象 $P(D_P,B_P)$ 满足 $D_P\in[\mathrm{Max}(0,D_Q-r),\mathrm{Min}(D_Q+r,R)]$。

由定义 6 可以看出,在不需要高维向量距离计算情况下,根据点对象的位码索引值就可以判断该点是否属于候选点,从而降低高维计算代价,有效克服“维数灾难”影响. 候选分区环内的点对象即为最终的 KNN 查询结果候选点对象. 以二维空间为例,假设两个查询 $\boldsymbol{Q}_1$ 和 $\boldsymbol{Q}_2$,$\boldsymbol{Q}_1$ 查询的候选分区为数据簇 B 的 A_3 区和数据簇 D 的 A_2 区(图 5 中分别用深灰和灰色表示),$\boldsymbol{Q}_2$ 查询的候选分区为数据簇 C 的 A_1 区(图 5 中浅灰表示),各自对应的候选分区环如图 6 所示.

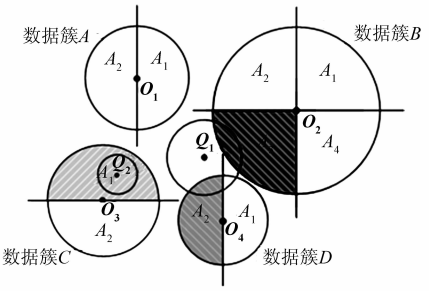

图5 候选分区

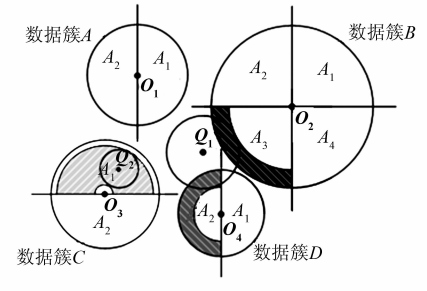

图6 候选分区环

4.2 KNN 查询

我们设计了 MapReduce 框架下基于 $G-L$ 索引的

并行 KNN 查询方法,以图 6 中的 Q_1 查询为例,KNN 查询过程如图 7 所示.

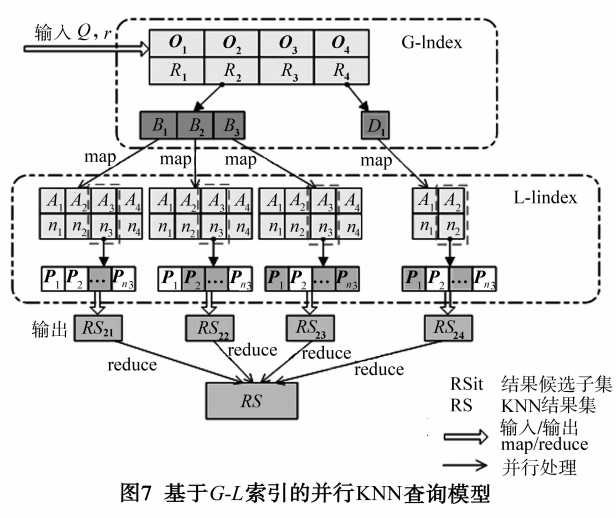

图7 基于*G-L*索引的并行KNN查询模型

算法 1 (候选数据簇筛选) $FilterCluster(\boldsymbol{Q}, G-Index, r)$

输入:查询点 $\boldsymbol{Q}$,全局索引 $G-Index$、查询半径 r

输出:候选数据簇列表 $CanCList$

步骤 1:对 $G-Index$ 中的每个数据簇 $C_i<\boldsymbol{O}(o_{i1},o_{i2},\cdots,o_{id}),R_i>$,确定 Q 的位码索引值 $\boldsymbol{Q}(D_Q,B_Q)$:

①若 $D_Q<R_i+r$,则 C_i 为候选数据簇,将 $<C_i,\boldsymbol{Q}>$ 存入 $CanCList$,并按 D_Q 升序排序.

②否则,数据簇 C_i 被过滤掉.

步骤 2:若 $CanCList$ 为空,则 $r=r+c$(c 为半径增长步长),并返回步骤 1. 否则,输出 $CanCList$,算法结束.

算法 2 (KNN 并行查询算法) $KNN(\boldsymbol{Q},r,G-L,k)$

输入:查询点 $\boldsymbol{Q}$、查询范围 r、双层索引 $G-L$、最近邻 k

输出:$\boldsymbol{Q}$ 的 k 最近邻

步骤 1:在 Master 节点提交作业时,执行:

①执行 $FilterCluster(\boldsymbol{Q},G-Index,r)$ 算法,得到候选簇列表 $CanCList=\{C_s,C_t,\cdots,C_w\}$;

②查询定位表,确定候选簇所在的 Slave 节点;

③根据全局索引,对 Slave 节点上对应的每个 block 分配 Map 任务;

步骤 2:在 Map 阶段,并行执行:

①根据局部索引,对每个候选簇 C_i,确定 $\boldsymbol{Q}$ 的候选分区集合 AS;

②根据定义 6,确定每个候选分区的候选分区环;

③将候选分区环内的点对象输出到候选结果子集 $RS_{it}=\{<\boldsymbol{Q},\boldsymbol{P}_s>,<\boldsymbol{Q},\boldsymbol{P}_u>,\cdots,<\boldsymbol{Q},\boldsymbol{P}_w>\},\boldsymbol{P}_s,\boldsymbol{P}_u,\cdots,\boldsymbol{P}_w$ 为点对象信息,i 表示候选簇编号,t 表示 block 编号;

④将候选结果子集 $RS_{it}(1\leqslant i\leqslant m,1\leqslant t\leqslant T)$ 输入 Reduce 函数. 其中 T 代表存储数据簇 C_i 的数据块($block$)个数;

步骤 3:在 Reduce 阶段,执行:

①统计候选结果子集 RS_{it} 点对象数量 $\boldsymbol{N}$,产生候选对象集 $<(\boldsymbol{P}_s,\boldsymbol{P}_u,\cdots,\boldsymbol{P}_w,\cdots,\boldsymbol{P}_v),N>$;

②若 $N=k$,则输出 $\{\boldsymbol{P}_s,\boldsymbol{P}_u,\cdots,\boldsymbol{P}_w,\cdots,\boldsymbol{P}_v\}$,算法结束;

③若 $N>k$,则计算 $\boldsymbol{Q}$ 与 $\boldsymbol{P}_s,\boldsymbol{P}_u,\cdots,\boldsymbol{P}_w,\cdots,\boldsymbol{P}_v$ 的欧氏距离并按升序排序,返回前 k 个即为 $\boldsymbol{Q}$ 的 K 最近邻;

④若 $N<k$,则 $r=r+c$,执行步骤 1.

5 实验结果与分析

5.1 实验环境

实验由 4 台 HP 刀片服务器搭建了一个内部网络,组成 4 个节点的 Hadoop 集群,其中 1 个节点作为 Master,其余 3 个节点作为 Slave,网络内的各个节点通过 100M 网卡相互访问. Master 节点服务器 CPU:Inter(R) Xeon(R) E5620 2.4GHz 4 * 4 核,Memory:8GB,Disk:300G * 8. Salve 节点服务器 CPU:Inter(R) Xeon(TM) 3.00GHZ 4 核,Memory:1GB,Disk:146.8G * 2. 每台服务器上安装 OS:64bit CentOS6.2,Hadoop 版本 1.0.3 和 Eclipse 版本 4.3.1. Hadoop 默认参数配置 block 为 64M,数据备份数为 3.

由于没有合适规模的真实数据集,本文按照表 2 的数据格式生成了一批聚类数据,其中每类数据的基准维度为 20,所有值均为 0 - 10000 之间的整数,以 2000 万条记录为标准,大约 2G 的数据量.

表 2 数据格式

ID	属性 1	属性 2	…	属性 20
T0	23	568	…	6735
T1	2567	381	…	752
…	…	…	…	…
T19999999	56	3476	…	5427

MapReduce 框架下默认的分区划分是通过 HashPartitioner 类实现,HashPartitioner 继承的是 Partitioner 类,是 Mapper 作业使用 key 的 hashCode 对数据进行均匀划分的方法. 为了实现 iPBM 的分区划分,实验中利用 eclipse 向 MapReduce 框架源码中增加新的分区划分类,命名为 BitCodePartitioner 并继承 Partitioner 类,再修改 MapReduce 配置文件使默认分区方式指向 BitCodePartitioner,使 BitCodePartitioner 代替 HashPartitioner 成为默认分区划分类,将修改后的 MapReduce 源码利用 Ant Builder 进行重新打包编译成为新的 MapReduce 框架,即 iPBM 框架.

5.2 实验结果与分析

影响实验运行时间的因素有:数据规模、数据维度、查询 K 值、服务器数量等. 因此,在研究某一因素对执行时间的影响时,需要保证其他因素固定不变,以保证实验数据的可靠性. 由于本文中数据划分和索引构建是预处理过程,一次预处理可以为后续的多次查询提

供服务,因此预处理时间未计入总时间.

5.2.1 数据规模变化对查询时间的影响

图 8 展示了数据维度为 20 维,K = 20,集群节点数为 4,数据规模分别为 2 千万、4 千万、6 千万、8 千万、1 亿、2 亿条记录时,KNN 查询执行时间的变化情况. 从实验结果来看:当数据维度、K 值、服务器数量等固定时,执行时间随着数据规模的增加而近似线性增长,即本文方法对数据规模的变化不敏感. 通过分析可知 iPBM 设计的大规模高维数据预处理和索引机制,有效地抑制了查询时间随数据规模急剧增长. 聚类预处理将相关数据分布在同一 Salve 上,使得查询操作集中在 Map 阶段而非 Reduce 阶段,从而避免了网络传输代价;分区预处理使得分区数据均匀分配到多个 block 中,这样可以充分利用 Map 任务在集群上分布式计算和任务处理能力;同时 KNN 查询通过双层索引实现了三级过滤过程进一步缩小实际需要搜索的范围,因此数据规模变化对查询时间的影响不大.

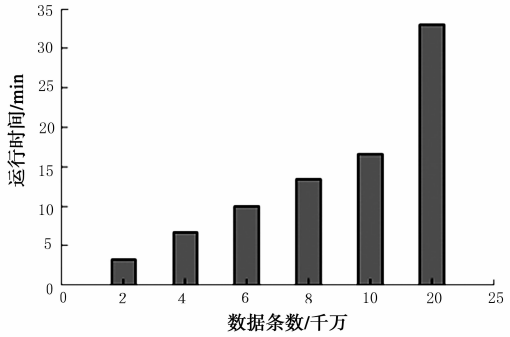

图8 数据规模对查询时间的影响

5.2.2 扩展性

从两个方面考察 iPBM 方法的扩展性:

(1)数据维度变化效果:图 9 展示了数据规模为 2 千万,K = 20,集群节点数为 4,数据维度分别为 10、20、30、40、50、100 时,KNN 查询执行时间的变化情况.

从实验结果来看:当数据规模、K 值、服务器数量等固定时,执行时间随着数据维度的增加呈上升趋势且上升的幅度相对稳定,基本保持在 43.7% 的水平,未出现随着维度的增加计算时间大幅增加的情况. 同时实验数据表明,随维度不断增大而查询时间增长放缓(从 10 维到 50 维,查询时间增长率分别为 71.3%,56.8%,18.8%,7.8%),说明 iPBM 对高维数据具有很好的抗压性. 在 iPBM 中使用了三层数据过滤技术,包括聚类、分区以及候选分区环,大大降低了数据搜索范围,同时利用数据位码索引值压缩表示形式,降低了高维向量距离计算代价,这是维度对查询时间影响不大的主要原因. 特别需要说明的是,实验中为了降低 MapReduce 框架下数据传输代价的影响,我们通过调整 Hadoop 参数配置(如表 3 所示),使得在不同维度下数据划分的 block 数量尽量保持一致,这样实验结果更能反映维度变化对查询时间的影响.

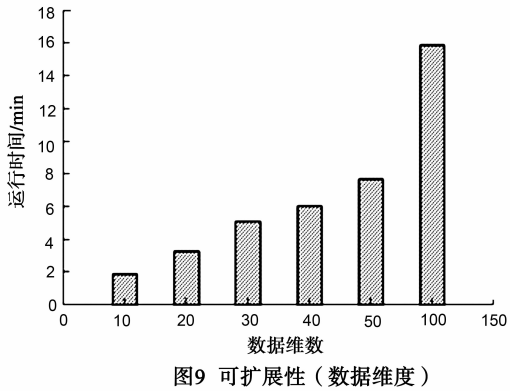

图9 可扩展性(数据维度)

表 3 不同数据维数下的 Hadoop 参数配置

数据维数	10	20	30	40	50	100
Block Size(MB)	64	100	150	200	256	512
Block Number(个)	16	21	21	20	20	20

(2)集群节点数量变化效果:图 10 展示了数据规模为 2000 万,维度为 20,K = 20,Slave 节点数分别为 3、6、9、12、15 时,KNN 查询执行时间的变化情况. 从实验结果来看:当数据规模、数据维度、K 值等固定时,执行时间随着服务器数量增加而减少的趋势很明显,平均降幅接近 1/3. 随着集群节点数量增多,通过 iPBM 方法将数据均分到更多的服务器上参与计算和任务处理,增大数据处理能力从而减少查询时间.

以上两个方面都说明了 iPBM 方法具有较好的可扩展性.

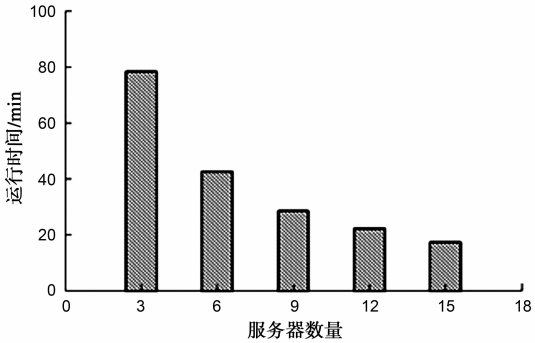

图10 可扩展性(服务器数量)

5.2.3 K 值变化对查询时间的影响

图 11 展示了数据规模为 2000 万,维度为 20,集群节点数为 4,K 值分别为 10、20、30、40、50 时,KNN 查询执行时间的变化情况. 从实验结果来看:当数据规模、数据维度、服务器数量等固定时,k 值的变化对执行时间基本没有影响. 分析可知 iPBM 在数据划分时将分区包含对象数量保持为 K 的数倍,使得每个分区的记录数

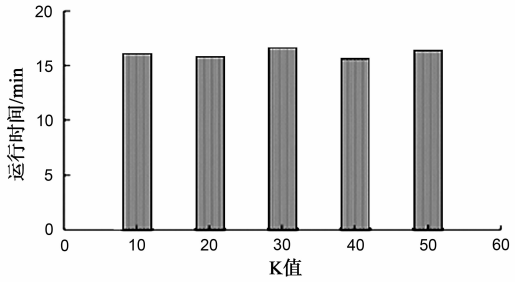

图11 K值变化对执行时间的影响

足够满足 KNN 检索结果的需要,因此不同 K 值筛选出的候选分区和对应的 block 集合基本一致,即不同 K 值需要计算的数据基本相同,从而 KNN 查询消耗时间差距不大.

5.2.4 iPBM 与非优化的 MapReduce 方法对比

(1)高效性:由于 iPBM 是在 MapReduce 框架下对数据划分做了优化处理,同时利用分布式双层索引提高查询效率. 为了体现这些策略的优势,我们将 iPBM 与传统 MapReduce 模型的查询效率进行对比分析. 图 12 展示了数据维度为 20 维,K = 20,集群节点数为 4,数据规模分别为 2 千万、4 千万、6 千万、8 千万、1 亿、2 亿条记录时,iPBM 与未经任何优化处理的 MapReduce 模型的性能对比. 从实验结果来看,同等条件下 iPBM 查询花费时间只有传统 MapReduce 模型的 45.6% 左右,效率提升的效果非常明显. 充分说明 iPBM 采用的兼顾相关性和并行性的数据分块方式,较 MapReduce 默认的数据分块方式,能够更好地适应大规模数据应用环境. 同时也说明了 iPBM 将高维向量转换为二维位码索引值降低了高维距离计算复杂性,以及双层索引机制大大缩小了搜索范围,这些策略对提升查询效率发挥了重要作用.

(2)准确性:iPBM 是在 MapReduce 框架下修改了数据划分方式,为了验证新的划分方式下 KNN 查询结果的准确性,我们将 iPBM 与传统 MapReduce 模型进行了对比分析,图 13 展示了数据维度为 20 维,K = 20,集群节点数为 4,数据规模分别为 2 千万、4 千万、6 千万、8 千万、1 亿、2 亿条记录时两者的准确度. 由于非优化的 MapReduce 框架下 KNN 查询是对所有的数据进行全范围扫描,因此能够保障查询准确性. 而从实验结果来看,同等条件下 iPBM 准确性与非优化 MapReduce 相当,这是因为 iPBM 在 KNN 查询时既考虑了单一簇范围内的结果查询,也考虑了跨簇范围下结果查询,不存在范围漏查的可能性,因此 iPBM 能够保证较高的查询准确性.

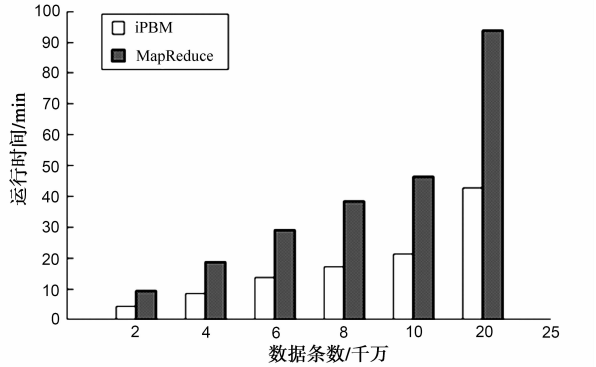

图12 iPBM与非优化的MapReduce的效率对比

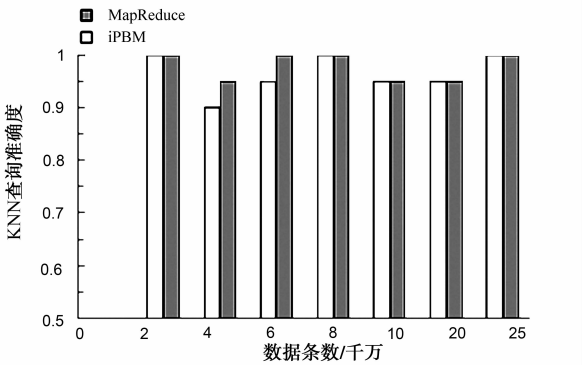

图13 iPBM与非优化的MapReduce的准确度对比

6 结束语

针对 MapReduce 数据和任务并行处理机制,提出了一种适用于 MapReduce 框架下的高维数据近似查询方法. 针对大规模高维向量空间分布特点、近似查询需求以及 MapReduce 编程模式,设计了基于聚类和分区的数据划分优化策略,将整个数据集均匀划分到集群中各个数据块(block)并共同承担计算任务,有利于充分发挥 MapReduce 任务并行处理优势;基于划分思想构建分布式双层索引,全局索引位于 Master 节点上用于 KNN 查询时对候选数据簇的筛选,局部索引位于 Slave 节点上用于进一步确定候选分区和候选点对象,实现三层过滤过程大大缩小需要搜索的范围;同时利用二维位码索引值压缩高维向量表示以降低维数灾难影响. 实验表明基于 MapReduce 编程模型的高维数据近似查询方法 iPBM 对查询效率具有明显的提升效果,同时具有良好的扩展性. 本文的研究工作主要是针对高维数据集比较稳定,索引一次构建多次使用的环境,没有考虑索引更新维护代价. 下一步需要研究在数据更新比较频繁的应用场景下,如何提升查询性能.

参考文献

[1] 黄震华. 云环境下 Top-n 推荐算法[J]. 电子学报,2015,43(1):54 - 61.
Huang Zhenhua. Top-n recommendation algorithms for cloud data[J]. Acta Electronica Sinica,2015,43(1):54 - 61. (in Chinese)

[2] 李建江,崔健,王聃,等. MapReduce 并行编程模型研究综

述[J]. 电子学报,2011,39(11):2635 – 2642.
Li Jianjiang,Cui Jian,Wang Dan,et al. Survey of MapReduce parallel programming model[J]. Acta Electronica Sinica,2011,39(11):2635 – 2642. (in Chinese)
[3] Dean J,Ghemawat S. MapReduce:simplified data processing on large clusters[J]. Proceedings of Operating Systems Design and Implementation ,2004,51(1):107 – 113.
[4] Zhang C,Li F,Jestes J. Efficient parallel kNN joins for large data in MapReduce[A]. Proceedings of the 15th International Conference on Extending Database Technology [C]. ACM,2012. 38 – 49.
[5] Doulkeridis C,Nϕrvåg K. A survey of large-scale analytical query processing in MapReduce[J]. Vldb Journal,2014,23(3):355 – 380.
[6] Vlachou A,Doulkeridis C,Nϕrvåg K. Distributed top-k query processing by exploiting skyline summaries[J]. Distributed & Parallel Databases,2012,30(3 – 4):1 – 33.
[7] Vlachou A,Doulkeridis C,Kjetil G,et al. On efficient top-k query processing in highly distributed environments[A]. ACM Sigmod International Conference on Management of Data[C]. ACM,2008. 753 – 764.
[8] 史英杰,孟小峰. 云数据管理系统中查询技术研究综述[J]. 计算机学报,2013,36(2):209 – 225.
Shi Yingjie,Meng Xiaofeng. A survey of query techniques in cloud data management systems[J]. Chinese Journal of Computers,2013,36(2):209 – 225. (in Chinese)
[9] Dittrich J,Quiané-Ruiz J A,Jindal A,et al. Hadoop ++:making a yellow elephant run like a cheetah (Without It Even Noticing)[J]. Proceedings of the Vldb Endowment,2010,3(12):518 – 529.
[10] Dittrich J,Quiané-Ruiz J A,Richter S,et al. Only aggressive elephants are fast elephants[J]. Infection,2012,5(11):243.
[11] Eltabakh M Y,Tian Y,Zcan F,et al. CoHadoop:flexible data placement and its exploitation in Hadoop[J]. Proceedings of the Vldb Endowment,2011,4(9):575 – 585.
[12] Zåschke T,Zimmerli C,Norrie M C. The ph-tree:A space-efficient storage structure and multi-dimensional index[A]. Proceedings of the 2014 ACM Sigmod International Conference on Management of Data[C]. ACM,2014:397 – 408.
[13] Wu S,Jiang D,Ooi B C,et al. Efficient B-tree based indexing for cloud data processing[J]. Proceedings of the VLDB Endowment,2010,3(1 – 2):1207 – 1218.
[14] Wang J,Wu S,Gao H,et al. Indexing multi-dimensional data in a cloud system[A]. Proceedings of the 2010 ACM Sigmod International Conference on Management of Data [C]. ACM,2010:591 – 602.
[15] Ding L,Qiao B,Wang G,et al. An efficient Quad-Tree Based Index Structure for Cloud Data Management[M]. Springer Berlin Heidelberg,2011. 238 – 250.
[16] 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.
[17] Doulkeridis C,Nϕrvåg K. On saying enough already in MapReduce[A]. Proceedings of the 1st International Workshop on Cloud Intelligence[C]. ACM,2012:7.
[18] 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.
Liu Yi,Jing Ning,Chen Luo,et al. Algorithm for processing k-nearest join based on R-tree in MapReduce[J]. Journal of Software,2013,24(8):1836 – 1851. (in Chinese)
[19] Liu Yi,Jing Ning,Chen Luo,et al. Parallel bulk-loading of spatial data with MapReduce:An R-tree case[J]. Wuhan University Journal of Natural Sciences,2011,16(6):513 – 519.
[20] 乔玉龙,潘正祥,孙圣和. 一种改进的快速 k-近邻分类算法[J]. 电子学报,2005,33(6):1146 – 1149.
Qiao Yulong,Pan Zhengxiang,Sun Shenghe. Improved K nearest neighbors classification algorithm[J]. Acta Electronica Sinica,2005,33(6):1146 – 1149. (in Chinese)
[21] 付宁,乔立岩,彭喜元. 基于改进 K-means 聚类和霍夫变换的稀疏源混合矩阵盲估计算法[J]. 电子学报,2009,37(4):92 – 96.
Fu Ning,Qiao Liyan,Peng Xiyuan. Blind recovery of mixing matrix with sparse sources based on improved K-means clustering and hough transform[J]. Acta Electronica Sinica,2009,37 (4):92 – 96. (in Chinese)
[22] 杨静,赵家石,张健沛. 一种面向高维数据挖掘的隐私保护方法[J]. 电子学报,2013,41(11):2187 – 2192.
Yang Jing,Zhao Jiashi,Zhang Jianpei. A privacy preservation method for high dimensional data mining[J]. Acta Electronica Sinica,2013,41(11):2187 – 2192. (in Chinese)

作者简介

梁俊杰 女,1974 年出生,湖北武汉人. 教授、硕士生导师、CCF 会员. 研究方向为大数据、高维索引、数据库管理系统.
E-mail:ljjhubu@163.com

李凤华(通信作者) 男,1966 年生,湖北浠水人,博士,中国科学院信息工程研究所副总工、研究员、博士生导师,研究方向为网络与系统安全、隐私保护、可信计算.
E-mail:lfh@iie.ac.cn