

PCPIR-V: 基于 Spark 的并行隐私保护近邻查询算法

邓诗卓, 姚继涛, 王波涛, 陈月梅, 袁野, 李艳辉, 王国仁

(东北大学计算机科学与工程学院, 辽宁 沈阳 110819)

摘要: 针对面向大数据的隐私保护查询效率低问题, 利用 CPIR 保护程度高, 实现了基于 Spark 的并行 CPIR 空间近邻查询隐私保护算法 PCPIR-V, 提出了基于 Row 和 Bit 的并行策略, 同时提出并实现了基于聚类的 PCPIR-V 的缓存优化技术。利用均匀分布、高斯分布和真实数据对 PCPIR-V 进行了测试验证, 在 40 个核心范围内, PCPIR-V 具有良好的扩展性, PCPIR-V 缓存优化技术计算时间与朴素 PCPIR-V 时间相比, 平均减少了 20%。

关键词: 查询隐私保护; 基于计算能力的隐私信息检索; Spark; 基于位置服务

中图分类号: TP309

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2016.00057

PCPIR-V: parallel privacy protected algorithms for nearest neighbor query based on Spark

DENG Shi-zhuo, YAO Ji-tao, WANG Bo-tao, CHEN Yue-mei, YUAN Ye, LI Yan-hui, WANG Guo-ren

(College of Computer Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: To address the low-efficiency problem for query privacy protection on big data, parallel CPIR-V (PCPIR-V), which had a high level of privacy protection for nearest neighbor query, was presented and implemented based on spark. Two parallel strategies for PCPIR-V, Row strategy and Bit strategy, were proposed. To avoid redundant multiplications, the repeated products were cached based on a clustering technique while computing CPIR on Spark. According to the evaluation results of PCPIR-V on three datasets, the scalability of PCPIR-V is good until the number of core is larger than 40. The cost of PCPIR-V with the method of caching partial multiplication results is reduced by 20% averagely.

Key words: query privacy protection, computational private information retrieval, Spark, location based service

1 引言

科学技术和互联网的快速发展促使数据爆炸式增长, 在大数据时代, 每天的生活和生产环境都会产生大规模海量的数据信息, 其中, 伴随着多样定位手段、用户终端及广泛通信手段的出现, 基于位置的服务^[1] (LBS, location based services)

已经成为当前研究的热点并且产生了大量基于位置的空间数据。LBS 已经深入到人们的日常生活中, 例如, 城市基础交通设施优化计划和基于用户行走路线的智能商店布局决策、智能路线导航 (如 Google Map 等)、基于位置的社交服务 (如 WeChat、Foursquare 等)、基于位置的游戏和娱乐 (如 Google Ingress 等)。随着 LBS 的用户大量增

收稿日期: 2016-03-18; 修回日期: 2016-04-25。通信作者: 王波涛, wangbotao@cse.neu.edu.cn

基金项目: 国家自然科学基金资助项目 (No.61173030, No.61272181, No.61272182, No.61332014, No.61370154, No.61332006)

Foundation Item: The National Natural Science Foundation of China (No.61173030, No.61272181, No.61272182, No.61332014, No.61370154, No.61332006)

加, 用户在享用位置服务的同时, 其在互联网上的行为数据存在泄露的风险。随着用户对个人隐私需求的提高, 用户希望在发起位置查询从而获得服务的过程中, 用户查询信息的隐私能够得到保护, 因此, 基于用户隐私保护的大数据需求亟需实现。

单点位置信息的隐私查询保护的典型手段主要分为3类: 干扰技术^[2~4]、区域覆盖(隐匿空间技术)^[5~8]、基于密码学的保护方式。干扰技术和区域覆盖都是通过损失可用性或隐私性的方式来实现的, 而密码学的隐私保护方式, 可以比较好地达到可用性和隐私性的平衡, 密码学的隐私查询保护技术较有代表性的是隐私信息检索(PIR, private information retrieval), 密码学的保护方式大都为基于复杂计算的数学方式, 隐私保护强度高, 但是计算量十分巨大。基于隐私信息检索的研究主要分为两大类, 分别是基于信息理论的私有信息检索(IT-PIR, information theoretical private information retrieval)^[9]和基于计算的私有信息检索(CPIR, computational private information retrieval)^[10]。CPIR的实现基于数论中的同余理论及相关的研究成果。求解二次剩余(QR, quadratic residue)问题的难度与因子分解难度相当, 是其隐私保护的理论基础。CPIR假设攻击者(服务器)计算求解能力为多项式限制(polynomially bounded), 在这个前提下, 保证攻击者不能通过区分QR与二次非剩余QNR来区别用户对不同数据项的访问。

如图1所示, CPIR待查询数据库^[11]是一个 4×4 的数据矩阵 M , $M_{a,b}$ ($1 \leq a \leq L, 1 \leq b \leq L$) 表示矩阵 M 中的某个元素, 矩阵的每个元素为用户所要查询的内容, 查询的过程是, 移动用户首先随

机产生2个大质数 q_1, q_2 , $N = q_1 \times q_2$, 根据查询内容 x_{10} 位置信息生成包含 N 的二次剩余与二次非剩余集组成查询信息 (y_1, y_2, y_3, y_4) , 其中, $y_3 \in QNR$ 二次非剩余, 其余为二次剩余, 服务器会利用式(1)计算 M 中每一行 z_k 值, 其中, $w_{k,i}$ 由式(2)和式(3)计算, 式(2)是根据二次剩余相乘基本性质得出的。

$$z_k = \prod_{i=1}^N w_{k,i} \quad (1)$$

$$w_{k,i} = \begin{cases} 1, & M_{k,i} = 0 \\ y_i, & M_{k,i} = 1 \end{cases} \quad (2)$$

$$w_{k,i} = \begin{cases} y_i^2, & M_{k,i} = 0 \\ y_i, & M_{k,i} = 1 \end{cases} \quad (3)$$

服务端返回 (z_1, z_2, z_3, z_4) 给客户端, 客户端通过公式欧拉判别式(4)得到 R 值, 若 R 为真, 则 z_a 为二次剩余, $M_{a,b}$ 为0; 反之, $M_{a,b}$ 为1。

$$R = \left(z_a^{\frac{q_1-1}{2}} = 1 \mod q_1 \right) \wedge \left(z_a^{\frac{q_2-1}{2}} = 1 \mod q_2 \right) \quad (4)$$

由于服务器只拥有 N 的信息, 并没有 q_1, q_2 的信息, 从而达到保护客户端隐私的目的。图1为1 bit数据查询, 多比特可以通过多次相同计算以支持多种类型的数据隐私查询, 包括空间数据。

近年来, 基于PIR的近邻查询方面, 文献[12]针对大量兴趣点(POI, points of interest)查询代价问题, 提出adaptive query plan, 至少保证预定比例的查询结果正确, 但是adaptive query plan依赖真实用户行为与之前发布查询的阶段性变化统计, 为了避免违反保密性, 结合 ϵ -differential privacy和PIR构成计算框架; 基于PIR的关键词搜索方面, 文献[13]中提出一种机制KSPIR,

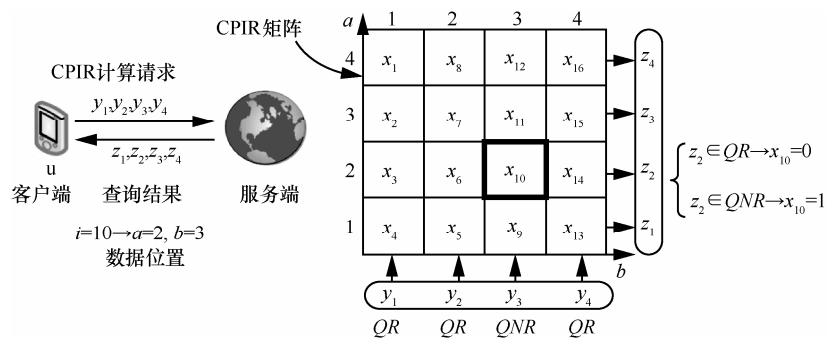


图1 CPIR 查询示例

通过用户指定的弹性隐私以及预算来最小化通信和计算代价, 该方法是检索代价和隐私强度 (degree) 之间的一个折中, 实现关键字检索; 文献[14]利用伪随机数规则随机生成排列数序列, 伪随机数加密规则能够模糊和加密查询结果。

云计算是大数据处理的技术支撑, 如分布式计算框架 Hadoop^[15]、Spark^[16]、Storm 等。Hadoop 中 MapReduce 侧重于离线数据分布式批处理, Storm 侧重于实时流分布式计算, 而 Spark 侧重基于内存计算, Spark 的快速发展和使用, 进一步提高了大数据处理的效率, 特别是在迭代计算和快速计算上相对于 MapReduce 有很大程度的提高。随着数据量增加, 大量用户希望得到在线隐私保护, 针对面向大数据的高强度基于位置服务的隐私保护问题, Spark 是在线的并行计算框架, 实现基于 Spark 的分布式 CPIR 位置服务隐私保护技术具有研究意义与应用价值。

在关于 CPIR 的分布式并行计算中, 主要实现分为 2 种: 基于 Peer-to-Peer 的并行化计算、基于主从结构 (master-slave) 的分布式计算。文献[17]提出一种基于 P2P 的并行 CPIR, 利用 striping technique 传输数据库给合作的 peer, 提高了 CPIR 查询的处理速度。文献[18,19]利用 MapReduce 计算框架实现 PIR 分布式计算。文献[18]中 PIRMAP 是一种适用于 MapReduce 的、有效的单服务器 CPIR 协议, 主要用于检索云上的大文件 (large files)。pCloud 是基于 GR-PIR 协议实现数据库分区并行的; PIRMAP 和 PRISM 对文件进行划分以实现并行化, 属于 I/O 级的并行化, 可以直接应用于 CPIR 计算的并行; 本文的并行化是对 CPIR 矩阵行 (Row) 计算以及列 (Bit) 计算的并行, 是 CPU 级的并行化, 与上述研究成果相比, 是 2 个不同维度的并行化, 两者之间是互相补充的关系。

2 CPIR-V 最近邻隐私保护查询算法

最近邻 (NN, nearest neighbor) 查询是 LBS 中一个应用比较广泛的查询, 获得查询对象距离最近的对象, 距离计算采用欧几里德距离。Voronoi 图^[20]通过对空间的划分体现空间对象之间的近邻拓扑关系, 被广泛应用到各个方面, 包括计算机图形学、空间数据关系、图像处理和多维数据等。图 2(a) 为 $p_1 \sim p_5$ 划分形成的 Voronoi 图^[1,11],

其中, 每个多边形称为 Voronoi 格, Voronoi 格的边为相邻空间对象的垂直平分线, 这样形成的多边形划分可以保证在同一个 Voronoi 格里面的所有空间对象的最近邻点都是同一个, 例如, 查询点 q 的最近邻点就是跟点 q 在同一个 Voronoi 格的点 p_2 。

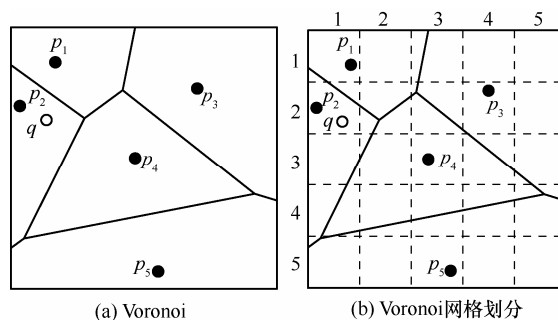


图2 Voronoi 图及其网格划分

文献[11]提出一种基于隐私数据检索和 Voronoi 图的最近邻隐私保护查询算法 CPIR-V, 如图 2(b)所示, CPIR-V 利用兴趣点 (POI, point of interests) 将 Voronoi 图进行网格划分, 实线是 Voronoi 边, 虚线显示的是经过划分的网格的边。 p 为 POI, q 为查询点, 查询点 q 在网格标号为 (2, 1) 的网格中, 该网格与 p_1, p_2 所形成的 Voronoi 格都有重合, 查询点 q 的最近邻可能是 p_1, p_2 中的一个, 所以, p_1, p_2 形成 q 最近邻的候选集合。不同的网格可能有不同数量的最近邻点, 这与点的分布、网格的大小都有关系。

构建网格矩阵, 以每个网格中 POI 数量最多为基准, 不足的网格进行填充。当用户发起查询的时候, 首先通过计算获取查询点所在的网格, 随机生成 2 个大质数 q_1, q_2 , 对 $N = q_1 \times q_2$ 进行二次剩余的计算, 生成图 3 中的查询 $q(y_1, y_2)$,

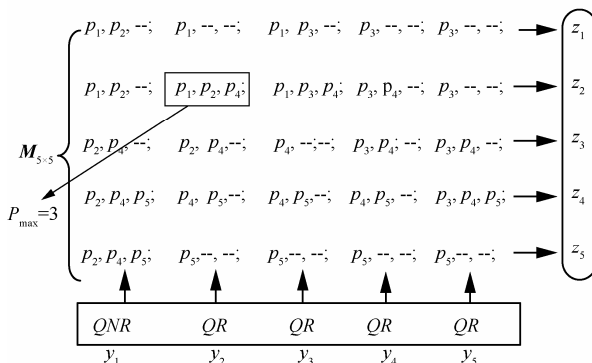


图3 CPIR-V 算法示例

y_3, y_4, y_5), 其中, 查询区域的网格对应的是 N 的二次非剩余, 其他位置是二次剩余, 即对矩阵第 2 行、第 1 列的查询信息的属性为 (QNR, QR, QR, QR, QR) , 利用 CPIR 计算 z 值。

3 基于 Spark 的最近邻查询算法 PCPIR-V

尽管 CPIR-V 实现了空间的最近邻隐私保护查询算法, 可以有效地保护用户的查询隐私, 但是, CPIR-V 算法有以下的不足。

1) CPIR-V 需要对整个矩阵进行扫描和计算, CPU 花费比较大。矩阵中存在许多重复大整数乘法运算, 计算量很大, 当数据库达到一定数量级的时候, 单机环境已经无法应对, 计算时间将无法满足用户的需求。

2) CPIR-V 算法由于空间分布的特性, 进行矩阵扫描和计算的时候有很多重复计算, 重复计算占用大量的计算资源, 从而降低了 CPIR-V 的计算效率。

并行计算的原理要求计算结果是可分解的且互相之间是没有影响的。文献[1,11]介绍了 CPIR-V 的并行计算原理: 基于 CPIR-V 的隐私保护近邻查询过程中, 每个 z 值的计算仅需要矩阵 M 中的一行数据和向量 y , 不同的 z 值结果互不影响。因此, 基于这一特性, 将矩阵 M 进行水平划分, 实现 CPIR-V 的并行计算。

因此, 本文采用 Spark 作为计算平台, 提出基于 Spark 实现并行的 CPIR-V 最近邻查询算法 PCPIR-V, 对 CPIR-V 基于 Spark 和 HBase 进行重新设计, 并根据数据的特性, 实现包括 Row 级并行和 Bit 级并行 2 种策略。基于 Row 的并行策略有数据分布不均匀的情况, 特别是在网格划分过小的情况下无法充分利用集群的资源进行计算, 造成集群资源的浪费, 针对此问题, 提出了

一种基于更加细粒度的并行策略——基于 Bit 的并行策略。

3.1 数据组织格式

通过对 HBase RowKey 的设计达到数据在 HRegionServer 上的均匀分布, 其中, 数据存储的格式为每个 RowKey 对应 PCIR-V 矩阵中的一行, RowKey 以逆序的行号对齐存储, 列名按照列号名称对齐存储。具体如表 1 所示。

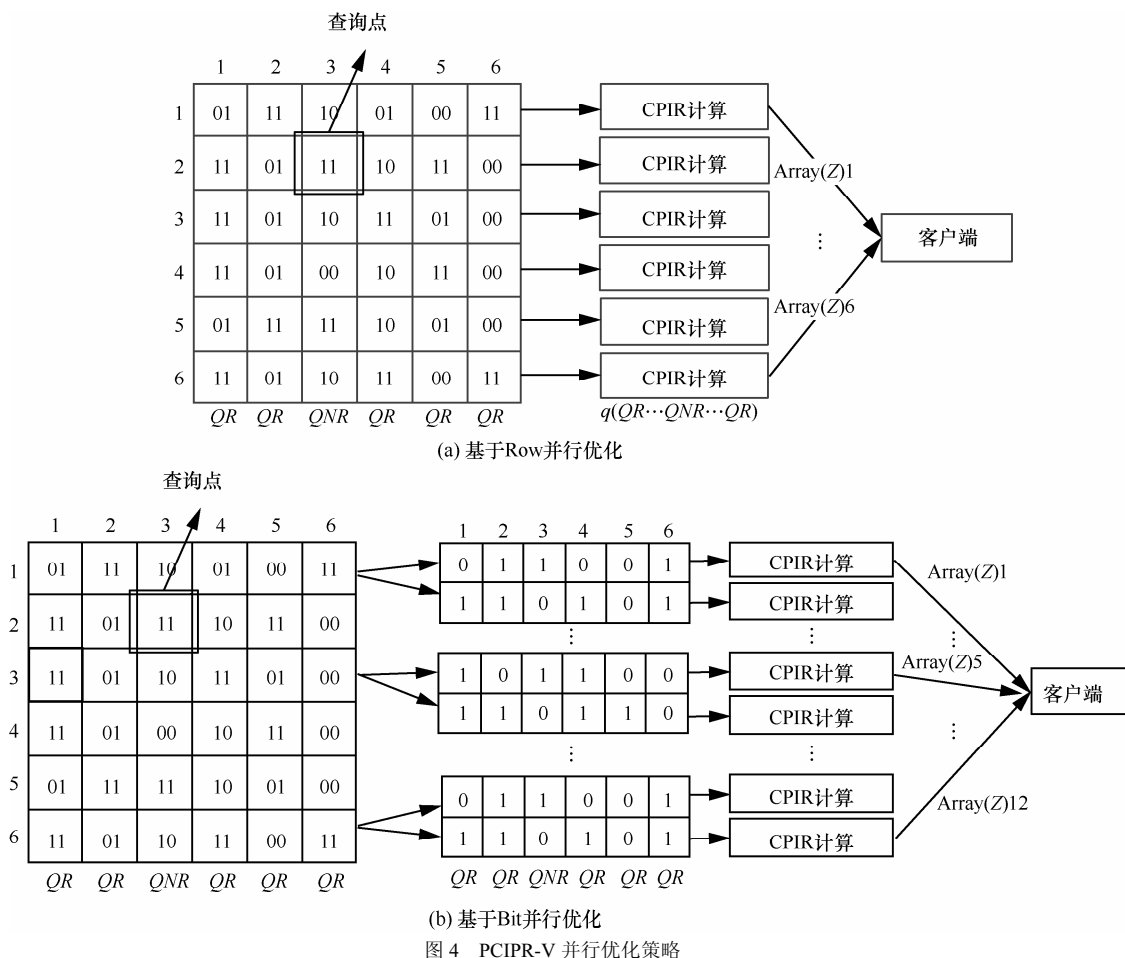
3.2 并行策略

PCPIR-V 采用 2 种并行的策略, 分别为基于 Row 的并行策略和基于 Bit 的并行策略。基于 Row 的并行策略是将数据中的每一行交给 Spark 的一个 Task 处理, 图 4(a)是基于 Row 的并行策略的 PCPIR-V 算法, 左侧 CPIR 矩阵每个网格存储 2 bit 的信息, 实际情况远不止这些, 可能是几百个或者几千个双精度的经纬度数值。PCPIR-V 算法将每一行的数据均单独交给 Spark 的一个 Task 进行 CPIR 计算, 将单机 CPIR-V 算法并行化, 提高了计算的效率, 并最终把每行的结果返回到客户端。但这种并行策略在网格内数据量较多的情况下, 由于每一行是并行的最小单位, 导致有些 Task 没有分配到任务, 因此, 存在集群计算资源分布不均的情况, 无法充分利用集群的计算资源。

基于 Bit 的并行策略通过将每一行按照 Bit 进行再次分组, 并按照当前的集群资源的状况 (包括当前 Executer 的个数和每个 Executer 最多运行的 Task 的个数) 确定分组的个数, 从而提高集群的利用率。图 4(b)是基于 Bit 并行策略的示意, 通过对每一行的内容继续细分, 组划分的个数可以通过集群现有资源进行确定。选取策略是将集群空闲的 CPU 核心数目作为分组个数, 每一组可能会被分到 1 个或多个比特, 图 4(a)中为 1 bit。

表 1 PCIR-V 信息表结构

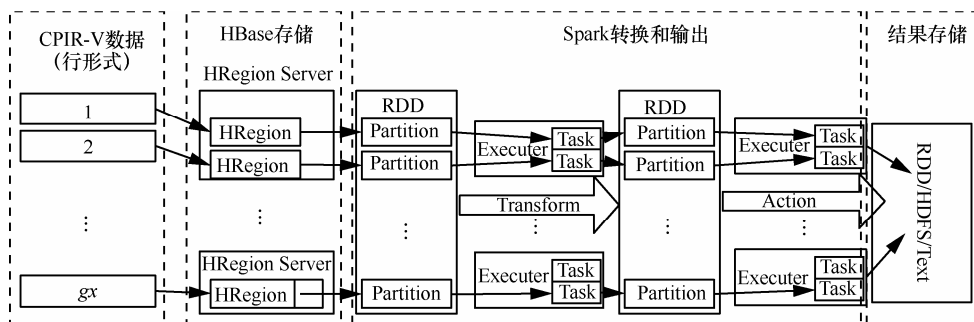
类型	名称	说明
RowKey	TimeReverse	按照行号逆序存储的一行键
Column Family Data	Column number_1	PCIR-V 矩阵当前行列号 1 中存储的潜在最近邻
	Column number_2	PCIR-V 矩阵当前行列号 2 中存储的潜在最近邻
	...	...
	Column number_n	PCIR-V 矩阵当前行列号 n 中存储的潜在最近邻



通过对计算的有效并行, 可以使 CPIR 计算的效率随着机器节点的增长, 性能不断提高, 且可以适用于大规模的数据量。PCIPR-V 不仅可以使数据均匀地存储, 而且可以将数据的 CPIR 计算进行均衡地分配, 特别是基于 Bit 的并行策略, 可以将并行的粒度更加细分, 节点的 CPIR 计算更加均衡。

PCIPR-V 算法读取 HBase 中的数据并缓存到 Spark 的分布式弹性数据集中, 由于数据在 HBase

中是分片存储的, PCIPR-V 会通过 RowKey 的设计策略使数据在集群中均匀地分配到各个 HRegion, 所以, 分片数据在弹性分布式数据集 (RDD, resilient distributed dataset) 中存储也被分散到各个 Partition, 通过 Spark 的 BlockManager 将 CPIR 计算分配到各个节点, 从而降低计算的时间, 提高效率。由图 5 可知, 通过对 RowKey 的设计将每行数据均匀地分布在各个 HRegion Server 的 HRegion 上, 之后 Spark 通过对 HBase



的读取将数据存放在 RDD 的各个 Partition 中供 Spark 进行 CPIR 计算, Executer 为每个 Partition 启动一个 Task 执行 CPIR 计算, PCPIR-V 的计算通过 RDD 的转换^[16] (Transform) 和动作^[16]

(Action) 2 步执行, 其中, Action 代表一个 Spark 任务的结束。如果在内存足够的情况下, Transform 是完全在内存中进行的, 相对 MapReduce 有较大的提高。PCPIR-V 的主要计算是在 Transform 中进行的, 最后通过 Action 输出结果。

3.3 PCPIR-V 算法

PCPIR-V 算法首先对文件中的空间位置信息进行读取, 生成 Voronoi 图, 然后, 通过 Voronoi 图和网格相交或者包含的位置关系, 求出每个网格的潜在最近邻点的矩阵存放到数组中, 之后将预处理的矩阵数据存放在 HBase 中, 数据通过 RowKey 设计被均衡分配到各个 HRegion 中, 并由各个节点的 HRegion Server 管理, 从而可以提高 Spark 从 HBase 获取的效率。

PCPIR-V 基于矩阵 Row 的并行策略和基于 Bit 的并行策略算法描述如算法 1 所示。

算法 1 PCPIR-V 服务端算法

输入: 查询 $q(y_1 \ y_2 \ \cdots \ y_n)$, 网格划分数目 g_x, g_y , 并行策略 s

输出: CPIR 计算结果 Array(Z)

1) 根据网格划分 g_x, g_y 获取 CPIR 矩阵数据缓存到 RDD;

2) if($s=Row$) then

3) 对 RDD 中的 CPIR 矩阵数据按照行进行分组;

4) end if

5) if($s=Bit$) then

6) 获得当前集群分配的 CPU 数量 k ;

7) 将每一行数据分为 k/g_y 组, 即总共 k 组;

8) end if

9) Spark 对每个分组进行 CPIR 计算获得 Z;

10) Spark 将结果 Z 聚合发送到客户端;

11) 结束。

算法 1 中, 步骤 1) 首先通过客户端传来的参数 g_x, g_y 计算出 HBase 中存储的行和列的最大值, 并通过 RowKey 最大值对 HBase 进行扫描获得矩阵, HBase 每一行作为 RDD 的一条记录; 步骤 2)~4) 是算法按照给定方式进行分组, 基于 Row 的并行策略是按照每一行对数据进行分组;

步骤 5)~8) 是策略为 Bit 时的分组情况, 首先, 根据当前集群分配计算资源的数量 k , 对每一行进行分组, 分组方式为取模 n , n 为查询序列的个数, 主要是通过 Spark 的 flatMap 和组数量的参数 k 对每一行的内容进行分组并形成新的条目; 步骤 9)~10) 通过 Spark 的 map 对每个小分组进行 CPIR 计算, 并将每一组的结果返回。

3.4 PCPIR-V 算法性能分析

由于 CPIR 算法需要对整个数据空间进行扫描和计算, 因此, 计算代价和总的数据量成正比。PCPIR-V 算法通过采用 Spark 集群技术将 CPIR-V 进行重新设计, 通过 HBase 的高速列存储和 Spark 基于内存的并行计算提高 CPIR 计算的速度, 并且拥有很高的扩展性。PCPIR-V 的服务端代价模型可用式(5)表示。

$$C = CIO + CT + CA \quad (5)$$

其中, C 为 PCPIR-V 算法的总代价, CIO 为从 HBase 中读取的 IO 代价, 此代价可以通过 RDD 的 cache 功能减免, CT 为 Spark 的 RDD 进行转换的时间, CA 是 Spark 的 RDD 进行动作的时间。其中, CIO 与 HBase 集群的数量、数据的分布和网络都有关系, CIO 随着查询数量的增大而增长, 并随着 HBase 集群规模的增大而减小, 而 CT 和 CA 均随着节点的 CPU 核心数的增多而降低, CT 和 CA 可以进一步用式(6)和式(7)表示, 其中, c 为 CPU 的核心数量, 代表时间随 CPU 增大而减小的系数。

$$CT = \beta \frac{Calct}{c} \quad (6)$$

$$CA = \beta \frac{Calca}{c} \quad (7)$$

3.5 基于 Spark 的 CPIR-V 的缓存优化

CPIR-V 算法由于空间分布的关系^[1], 相邻网格的潜在最近邻点存在较多的冗余点, 因此, 在进行 CPIR 计算的过程中, 有较多重复的计算和比较。

图 3 体现出了 CPIR-V 相邻的网格中存在较多重复的潜在最近邻, 这是 CPIR-V 的空间对象分布相邻导致的。当重复过多的时候, CPIR 的计算就会出现很多冗余计算, 因此, 对数据的预处理十分重要, 需要对数据进行精简和缓存策略来提高数据的计算效率。PCPIR-V 算法由于是 CPIR 基于 Spark 的并行化设计的方法, 所以, 存在同样的问题, 虽然通过并行使问题得到了解决, 但

还是存在冗余计算及计算资源的浪费,因此,需要对计算数据进行预处理和基于 Spark 的特性进行优化,从而减少冗余计算。图 6(a)展示了 CPIR 基本的计算方法,按照式(1)的计算方式, Z 的计算结果为 $Z_1=y_1 \times y_2 \times y_3 \times y_4 \times y_5$, $Z_2=y_2 \times y_3 \times y_6$, $Z_3=y_2 \times y_6$, $Z_4=y_1 \times y_2 \times y_3 \times y_4$, $Z_5=y_2 \times y_3 \times y_6$, $Z_6=y_2 \times y_6$, 可以看出 $y_2 \times y_3$ 计算了 4 次, $y_2 \times y_6$ 计算了 4 次, $y_1 \times y_2 \times y_3 \times y_4$ 计算了 2 次, $y_2 \times y_3 \times y_6$ 计算了 2 次,即存在很多冗余计算,如果每次都重复地计算,由于是大整数的乘法,将急剧消耗集群的计算资源。因此,需要对数据进行处理和采用缓存进行优化以解决冗余计算的问题。

3.5.1 PCIPR-V 缓存优化算法

不同于文献[1]中的关联规则方法,本文提出一种基于聚类 and 二进制交集的缓存优化方法,并通过 Spark 集群节点的特性将缓存分布式存储,提高计算速度。该方法将隐私查询分为 2 个阶段:数据预处理阶段和在线缓存查询阶段。其中,数据预处理阶段分为 2 个小的步骤: CPIR 数据聚类 and 缓存二元组生成。具体过程如下。

1) 将二进制的数组作为聚类的条目,如图 6(b)中的第一行的数据为 111110 序列,利用 k 均值聚类算法对数据库中所有的二进制数组进行聚类,对于这种二进制 0 或 1 的数据,本文提出一种优化距离和聚类中心的计算方式,从而适用于将位值大部分相同的记录聚集到一起的情况,为了适用于二进制 0 或 1 形式的聚类,设计一个新的距离和聚类中心的计算方法,新的计算方式如式(8)、式(9)所示。假设通过公式将类别分为 k 类,其中, a, b 代表数据矩阵中任意 2 行数据的二进制数组。 $round$ 为四舍五入函数, e 为某类中的一列中的二进制的数值, i 代表列数, $n_{cluster}$ 为本类的元素个数, c_i 代表每一列的平均值,则 $(c_1, c_2, \dots, c_{gy})$

为这一类的中心,其中, gy 为 CPIR 矩阵的列数。以 $a=111110, b=111100$ 为例,式(8)计算如下,首先,计算 $a \wedge b=111100$, $count(a \wedge b)=4$, 则 $d_{a,b}=\frac{1}{4}$ 。若 a, b 为同一类,且本类中只有 a, b 2 个元素,则 c_i 计算为 $c_i = round\left(\frac{1+1}{2}\right)=1$, 其他 c 的计算方式类似。

$$d_{a,b} = \frac{1}{count(a \wedge b)} \quad (8)$$

$$c_i = round\left(\frac{\sum_1^{gy} e_j}{n_{cluster}}\right) \quad (9)$$

通过 k -均值聚类算法,用式(8)、式(9)的距离公式和聚类中心计算公式对数据库所有的数据聚为 k 类, k 的取值与集群节点个数相同,这样就能保证相似度较高的二进制序列都在一类,并为每一类都标记上标签。

2) 分别对每一类中的所有数据相互进行判断,满足式(10)的条目添加到本类待缓存的 map 中, id 为 id_i , 其中, θ 是一个阈值,表示 2 条记录 a, b 同时为 1 位数的最小个数为 θ , 其中, θ 的取值规则为:满足 θ 个大整数相乘的时间大于一次内存查表时间条件的最小取值。如果满足下列条件则将 a, b 中 $a \wedge b$ 为 1 的部分置为 0, 分别变为 a', b' , 并将这 2 条记录分别以一个二元组的形式保存,如 $(a', id_a), (b', id_b)$, 其中, a', b' 叫作共享差, id_a, id_b 叫作共享基。如果每条记录产生了多个二元组,则选择 $count(a')$ 最小的进行保存。

$$count(a \wedge b) > \theta \quad (10)$$

3) 通过 Spark 将不同类别划分到不同节点上运行,并将每一类的待缓存数据优先计算,之后再对二元组进行计算,其中, id_a, id_b 所需要计算的值均可以从节点的本地缓存中获得,从而实现计

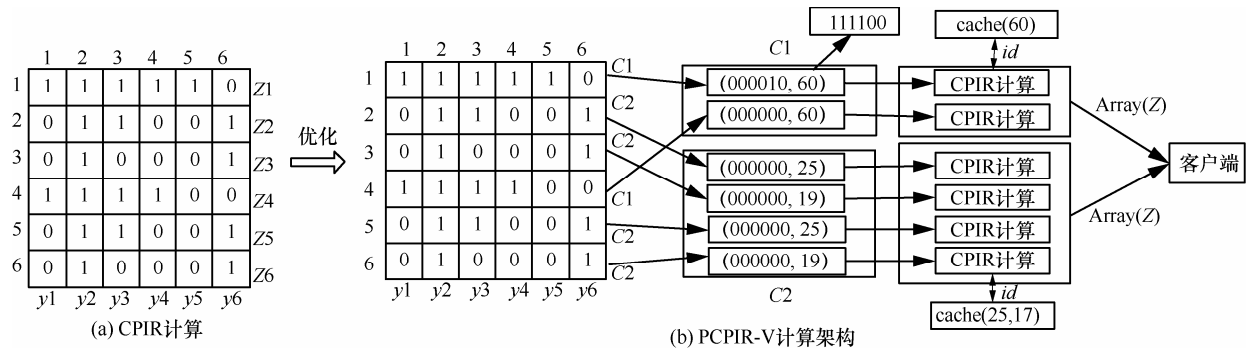


图6 PCIPR-V 优化算法架构

算的共享, 并且每个节点的运算都是并行执行的, 大大减少了计算代价, 提高了计算的效率。

图 6(b)的 PCPIR 计算框架展示了本优化算法的基本过程。首先, 将数据聚类, 采用式(8)和式(9), 将所有数据聚为 $C1$, $C2$ 两类, 然后将这两类分发到相同的节点并分别进行二元组转换, 对类内的每一条记录都互相进行判断, 如果满足式(10)的条件, 则生成一个新的二元组。二元组的第 1 个元素为共享差, 第 2 个位置为共享基, 共享基为 2 条记录取交后的编码, 图 6 中的 60 代表 111100 的编码, 代表 2 个 $C1$ 类的重复为 1 的位置为前 4 位, 其中, 如果一条记录有多条重复的情况, 则以 $count$ 最大的为准。共享差为将共享基为 1 的位置 0 所得的二进制数据, 例如, 图 6 中 $C1$ 类的第一行数据为 111110, 将 111100 为 1 的位置置 0 后为 000000。最后, 通过将不同类的二元组发送给 Spark 的不同节点进行执行, 首先, 对二元组的共享基进行计算, 放入到本节点的 cache 中, 之后, 再对二元组的共享差进行计算。由于共享差 0 的部分不需要进行大整数相乘计算, 而共享基的 id 位置已经存放到缓存中, 所以, 计算不仅可以分配到各个节点, 而且能提高计算效率并有效地减少重复计算。

算法 2 二进制 k -均值聚类算法

输入: 集群节点个数 k , CPIR-V 数据

输出: 类别标签

- 1) 随机选择 k 条数据作为初始聚类中心;
- 2) 对于每条数据使用式(8)计算与每个聚类中心的距离, 并将该条数据并入距离最近的中心;
- 3) 对于每一类按照式(9)重新计算聚类中心;
- 4) 如果聚类中心没有发生改变则结束, 否则跳到 2);
- 5) 结束。

算法 2 中, 步骤 1) 首先从数据集合中随机选择 k 条数据作为初始的聚类中心, 其中, k 的取值为 Spark 集群启动 Executer 节点个数, 每一类均有自己的缓存数据, 这样做可以将更加相似的记录聚集到一起, 缓存的命中率会更高; 步骤 2) 对所有数据与各类聚类中心进行距离的计算, 距离公式采用式(8)并将每条记录分配到距离最近的中心, 即式(9)取值最小的中心; 步骤 3) 对已经进行聚类完毕的类别分别计算聚类中心, 使用式(10); 最后, 步骤 4) 对收敛条件进行判断,

收敛条件为进行再次聚类的聚类中心较之前的中心没有改变。

算法 3 数据简化和待缓存数据生成算法

输入: 其中一类的数据 $data$, 数据的条数 n ,

阈值 θ

输出: 简化数据和待缓存数据的二元组 $tuple$

- 1) 声明临时数组变量 $temp$, $element$;
- 2) for ($int\ i=0; i<n; i++$)
- 3) for ($int\ j=0; j<n; j++$)
- 4) if ($i!=j$)
- 5) $temp[j]=data[i]^data[j]$
- 6) if ($count(temp[j]) > \theta$)
- 7) $element[j]=reverse(temp[j])$
 $^data[i]$;
- 8) end if
- 9) end if
- 10) end for
- 11) 从 $element$ 中选取 $count$ 最小的 $element$
 $[min]$, $tuple[i]=(element[min], encode(temp[j]))$;
- 12) end for
- 13) 结束。

算法 3 中, 步骤 2)~12)是对类内记录的相互判断; 步骤 4)是排除与本身比较的情况; 步骤 5)计算 2 条记录之间重复为 1 的结果; 步骤 6)按照式(8)进行判断, 满足才进行记录; 步骤 11)是在第一层循环完毕后, 从 $element$ 中选取一个重复为 1 最多的元素, 添加到二元组中, 同时二元组第 2 个元素为重复为 1 部分的编码。算法 3 对应图 6 中的第 2 个步骤预处理的数据可以通过 Spark 集群进行查询。

算法 4 缓存优化 PCPIR-V 查询服务端算法

输入: 查询 $q(y_1\ y_2\ \cdots\ y_n)$, 网格划分数目 g_x ,

g_y

输出: CPIR 计算结果 $Array(Z)$

- 1) 将二元组信息缓存到 RDD;
- 2) 对 RDD 中的二元组进行聚合, 获得(id , $List(value)$)的形式;
- 3) 通过查询 q 对(id , $List(value)$)中 id 对应的数据进行 CPIR 计算, 缓存到 RDD;
- 4) 利用缓存的数据和 q 计算 $List(value)$ 的 CPIR 结果, 并缓存到 RDD;
- 5) 聚合数据发回客户端;
- 6) 结束。

算法4中,步骤1) 首先将前2步生成的二元组数据读入到 Spark 的 RDD 中;步骤2) 按照缓存 id 对二元组进行聚合 (groupByKey) 生成一个新的二元组 ($id, seq(element)$), 这样保证 id 在 RDD 中是唯一的;步骤3) 则对 id 进行解码并同查询 q 进行 CPIR 计算得出结果,步骤4) 对剩下的 $element$ 相乘得到 ($seq(Z)$) 的序列。算法4展示了图6中 Spark 进行并行计算的过程: 首先, 对共享基缓存进行计算; 然后, 对整个数据进行计算; 最后, 对数据进行合并后将结果返回客户端。

3.5.2 PCIPR-V 缓存优化算法性能分析

PCIPR-V 缓存优化算法通过3个步骤解决了 PCIPR-V 中由于空间分布特性导致的过多冗余计算问题, 第1步对数据进行聚类, 是为了将相近的数据聚集在一起, 为第2个步骤提供方便; 只需要在类内比较就可以找到相似度最高的记录; 最后, 通过 Spark 对缓存数据进行优先计算, 并对缓存数据进行重用。

PCIPR-V 缓存优化算法的主要性能优化在于可以将各条记录内大整数乘法的计算进行缓存, 从而提高计算效率。缓存优化算法的代价模型可以用式(11)表示。

$$C = C_{\text{total}} - C_{\text{redundant}} + C_{\text{cache}} \quad (11)$$

其中, C 代表缓存优化算法的总代价, C_{total} 为没有缓存时的计算代价, $C_{\text{redundant}}$ 为重复记录的计算代价, C_{cache} 为缓存读取的代价。其中, C_{cache} 、 $C_{\text{redundant}}$ 的计算次数是一致的, 但缓存的读取时间要远远小于大整数相乘的运算时间, 因此, 可以起到对比较大的冗余计算的优化作用。其中, 缓存优化效果与数据的关系, 以及还有阈值 θ 的最优解问题将在未来的工作中进行研究和讨论。通过 Spark 分布式特性可以对这种缓存的计算进行并行, 更进一步提高计算的效率。

4 实验结果与分析

本实验采用的服务平台为 IBM xSeries System 3650M4, 集群共有5个节点, 每个节点详细配置如下。

CPU: 2×Xeon E5-2620 CPU (每个有6核心×2线程)

内存: 32 GB;

硬盘: 5 TB, 10 000 rpm, raid 5;

操作系统: CentOS 6.4;

开发工具: GNU Toolkits (G++、GDB)、Make、Vim, JDK 等。

实验所用开发语言为标准 C++、Java、Scala 语言, 使用合成数据和真实数据进行实验并对实验结果进行分析, 由于本文 PCIPR-V 与相关研究成果的并行化思路是不同维度且为互为补充的关系, 同时未发现同一 CPU 并行化维度的相关成果, 因此, 对比了 CPIR-V 和 PCIPR-V, 验证云环境下 PCIPR-V 的查询性能。合成数据采用生成均匀分布和高斯分布2种分布方式, 其中, 高斯分布满足 $(x,y) \times N(1,1,0,0,1)$ 。真实数据来自加利福尼亚的 Sequoia^[21]。本文的数据取值范围为 $1\ 046\ 435 \times 1\ 929\ 615$, 其中, x 、 y 坐标类型都为 int 型, 本文所指网格划分为 G , 是指将空间划分为 $G \times G$ 的均匀网格。本文 C++ 所使用的大数计算库为 GMP^[22], Java 为 JDK 自带的大整数计算工具。其中, CPU 核心数量指的是虚拟核心的数量, 即操作系统级别上的核心数量, 而非物理核心数量, 其中, 阈值的取值规则为满足 θ 个大整数相乘的时间大于一次内存查表时间条件的最小取值, 通过实验得出, 当 $\theta=3$ 时, 大整数相乘的耗时超过了内存查表时间, 因此, 本文 θ 取值为3。阈值 θ 的最优解问题将在未来工作中进行研究。

表2 PCIPR-V 实验数据参数

参数名称	变化范围	默认值
数据类型	真实数据 (6.2×10^4), 高斯分布 (1×10^5), 均匀分布 (1×10^5)	均匀分布 (1×10^5)
网格划分	10×10 , 20×20 , 50×50 , 100×100 , 200×200 , 400×400	100×100
模量 k	128, 256, 512, 1 024	512
范围查询覆盖率	1, 5, 10, 15	1
查询结果计算方法	式(2), 式(3)	式(2)
CPU 核心数量	1, 10, 20, 40, 80	80

4.1 PCIPR-V 不同数据服务端时间

图7为不同数据在不同网络划分下的服务端时间, 其中, 图7(a)~图7(c)是利用式(2)进行的网络划分, 图7(d)是利用式(3)进行的网格划分。图7(a)为基于 Spark 的并行 CPIR-V 算法 (PCIPR-V) Row 并行和 Bit 并行策略与未优化的 CPIR-V 算法在不同网络划分下使用不同数据服务端计算时间的对比, 其中, PCIPR-V-R 为基于 Row 并行策略的 PCIPR-V 算法, PCIPR-V-B 为基于 Bit 并行策略

的 PCPIR-V 算法。从图 7(a)中可以看出, 3 种算法都是随着网格划分的增大而变慢, 这是由于 CPIR-V 的矩阵列数与网格列数是一致的, 网格越多说明 CPIR-V 的矩阵越多, 需要相乘的大整数也就越多, 时间也就越长。在网格数目为 10 的时候, 3 种算法的计算时间基本是在同一个数量级上, 这是因为在网格划分太小的情况下, PCPIR-V 启动的任务是有限的, 不能很好地发挥并行的优势, 且在计算的过程可能会产生节点间通信的代价, 所以, 基于行并行的 PCPIR-V 算法在刚开始网格划分小的时候甚至超过单机的 CPIR-V 算法计算时间, 在对 PCPIR-V 进行 Bit 并行策略的优化之后, 可以充分利用集群资源进行并行计算, 在网格较小时, 相对基于行的 PCPIR-V 有很大的提升。随着网格数目的增长, PCPIR-V 相对传统的 CPIR-V 已经有明显的优势, 在网格数目在 400 的时候, 大概有 1 个数量级的提升, 且随网格增长, 时间增长的趋势比较平缓, 相对 CPIR-V 随网格增长计算时间增长较快。

图 7(b)为在高斯分布基于 Spark 的并行 CPIR-V 算法 (PCPIR-V) Row 并行和 Bit 并行策略对比。不同网格划分对高斯分布中最近邻查询影响的变化趋势与均匀分布相同, 但是高斯分布的服务端的计算时间相对于均匀分布有明显提高, 这是由于高斯数据中每个网格中潜在最近邻数量的最大值远远超过均匀分布, 从而导致计算数和返回的 QR、QNR 数目增大, 最终导致时间增大。

图 7(c)为在真实数据中数据中基于 Spark 的并行 CPIR-V 算法 (PCPIR-V) Row 并行和 Bit 并行策略与未优化的 CPIR-V 算法在不同网格划分下服务端计算时间对比图。虽然潜在最近邻的最大值很大, 但是潜在最近邻中的 0 bit 比较多, 所以, 计算时间相对均匀分布数据和高斯分布数据小很多。

图 7(d)展示了利用式(3)不同网格划分对均匀分布数据集最近邻查询的影响, 从图 7(d)中可以看出 3 种算法都随着网格数量的增加而增大, 基本的变化趋势和图 7(a)基本相同, 但是时间与图 7(a)相比都有所增大, 这是由于在计算矩阵中存在大量的 0 bit, 造成大量大整数的平方计算, 从而增加了计算代价。对于 CPIR-V 来说, 影响非常大, 而 PCPIR-V 就增长不明显, 仍可以保证良好的性能, 这是由于基于 Spark 的 PCPIR-V 算法

有比较高的扩展性, 通过将复杂的计算分配到各个节点, 从而达到高性能的计算水平, 所以, 当计算量急剧增大时, 对 CPIR-V 的影响很大, 但是对 PCPIR-V 却较小。

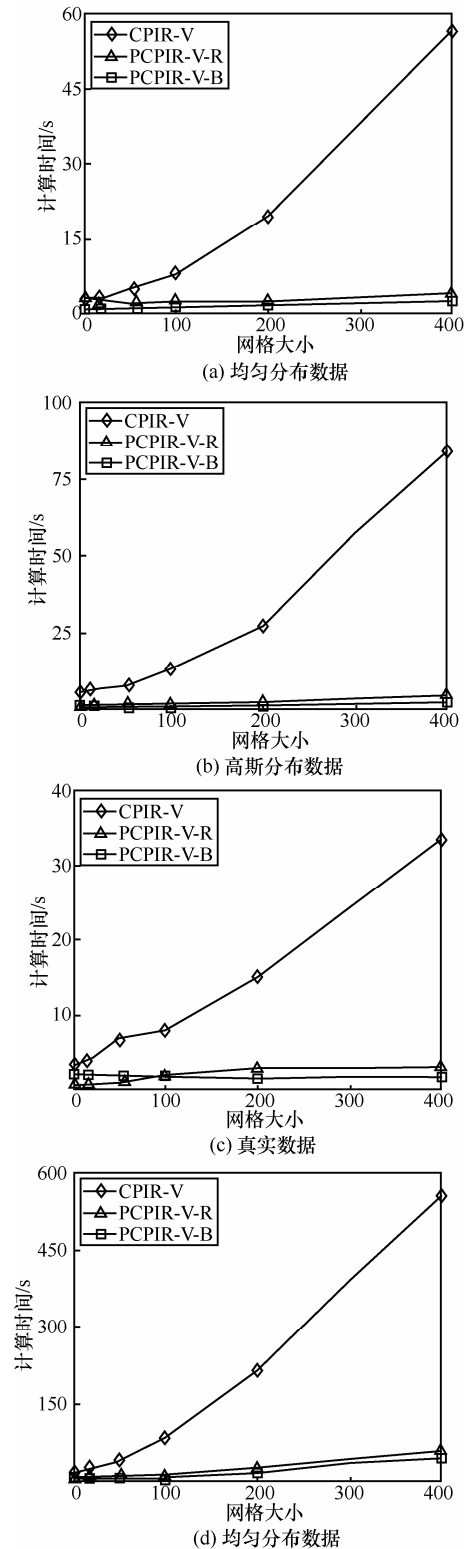


图7 不同数据在不同网格划分下的服务端时间

4.2 不同模量服务器时间

图8为基于Spark的并行CPIR-V算法(PCPIR-V) Bit并行策略与未优化的CPIR-V算法在不同模量下服务端计算时间对比。

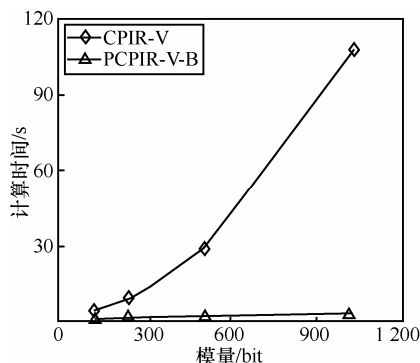


图8 不同模量的服务端时间

从图8中可以看出,随着模量的不断增加,服务端的计算时间也会随之增加,其中,PCPIR-V-B的增长速度较为平缓,而CPIR-V增长速度比较快。这说明在集群资源充足的情况下,模量的增长不会对PCPIR-V-B算法造成过大的影响。

4.3 不同CPU个数服务端时间

图9显示PCPIR-V算法在不同CPU核心数量下计算时间的变化,实验Spark集群共有5个服务器节点,应用到YARN集群的计算资源共有100个虚拟CPU核心。实验结果表明,随着CPU数量的增加,计算时间也在不断锐减。当达到一定程度后,速度开始变缓,这是因为CPU核心数量越多,算法的并行度就越大,计算时间就减少,但核心数量在达到一定程度时开始变缓,图9中核心数为40个,这说明在本组实验数据和实验条件下CPU核心数量超过40个的时候,主要瓶颈已经从CPU转到节点之间通信和交互。

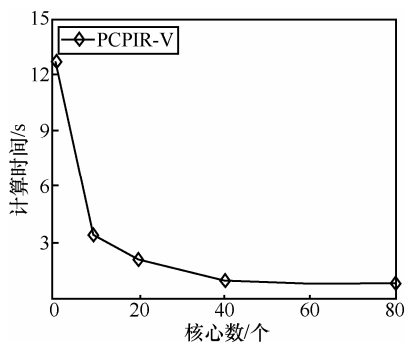


图9 不同核心数的服务端时间

4.4 基于Spark最近邻CPIR缓存优化服务端时间

本实验主要对服务端的算法做出优化,减少了数据的冗余计算,实现了计算共享。因此,只对服务端的算法做出实验分析,分别针对不同数据集和不同网格划分下的实验做出分析。

图10分别是缓存优化的PCPIR-V和PCPIR-V在均匀分布数据集、高斯分布数据集和真实数据集下,不同网格划分下的服务端计算时间的对比。其中,PCPIR-V-CA为基于缓存优化的PCPIR-V算法。

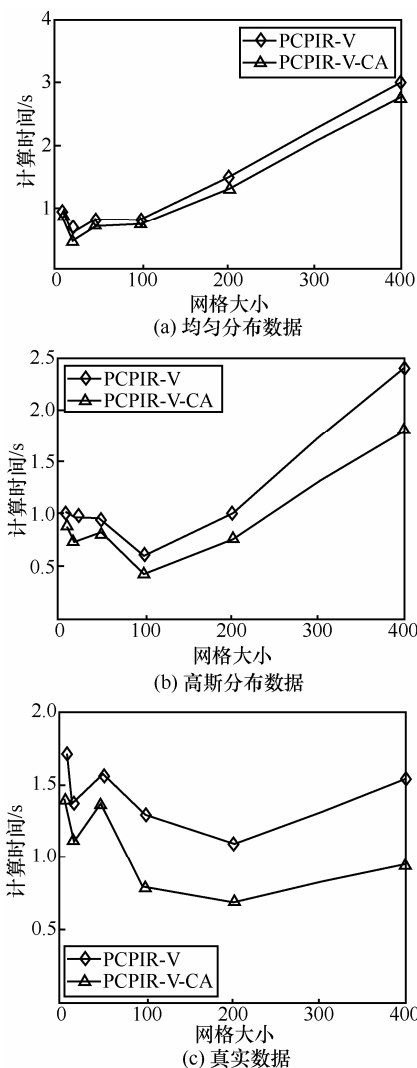


图10 不同数据在不同网格划分下的服务端时间

从3个实验结果对比来看,真实数据集和高斯数据集要比均匀数据集高出很多,这是由于高斯数据和真实数据的集中分布导致的,越集中分布的数据,可能存在的重复计算也就越多,这样缓存优化的效果也就越好。这2种

算法都会在网格刚开始的时候有一个较大的峰值, 之后降低, 这是因为在起初网格划分较小的时候, 网格内的潜在最近邻较多, 这样数据就不能很好地分配到各个节点分布式执行, 因此导致计算速度慢, 缓存的优化效果也不好。缓存的效果在网格划分小的时候不明显, 这是因为开始的主要瓶颈不是在大整数的计算上而是判断当前位是否为 1 上, 因此, 缓存的优化都是从网格数目在 100 之后才开始显著地提高, 特别是高斯分布的数据和真实数据, 主要是因为它们存在更多的重复计算。

5 结束语

针对面向大数据的隐私保护查询问题, 本文以 Spark 计算框架为基础, 利用 CPIR 高保护强度, 在 HBase 上设计了数据的组织格式, RDD 实现了并行 PCPIR-V 最近邻查询算法 (PCPIR-V), 并提出 2 种并行策略。针对 PCPIR-V 算法中冗余计算的问题, 提出了基于缓存的 PCPIR-V 算法, 通过使用聚类 and 二进制交集方法对数据进行了预处理, 减少了冗余计算, 提高了计算的效率。最后, 通过对实验进行分析, 在 40 个核心范围内, 基于 Row 和 Bit 并行策略的 PCPIR 具有良好的加速比, PCPIR-V 缓存优化技术计算时间与朴素 PCPIR-V 时间相比, 平均降低了 20%。

参考文献:

- [1] 王璐, 孟小峰. 位置大数据隐私保护研究综述[J]. 软件学报, 2014, 25(4):693-712.
WANG L, MENG X F. Position big data privacy protection research review[J]. Journal of Software, 2014, 25(4): 693-712.
- [2] NIU B, LI Q, ZHU X, et al. Achieving k -anonymity in privacy-aware location-based services[C]//2014 IEEE Conference on Computer Communications(Infocom). c2014: 754-762.
- [3] ZHOU C, MA C, YANG S, et al. A location privacy preserving method based on sensitive diversity for LBS[M]//Network and Parallel Computing. Berlin: Springer, 2014: 409-422.
- [4] CICEK A E, NERGIZ M E, SAYGIN Y. Ensuring location diversity in privacy-preserving spatio-temporal data publishing[J]. The International Journal on Very Large Data Bases, 2014, 23(4): 609-625.
- [5] PAN X, XU J, MENG X. Protecting location privacy against location-dependent attacks in mobile services[J]. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(8): 1506-1519.
- [6] ŠIKŠNYS L, THOMSEN J R, ŠALTENIS S, et al. A location privacy aware friend locator[M]. Berlin:Springer, 2009: 405-410.
- [7] CHOW C Y, MOKBEL M F, BAO J, et al. Query-aware location anonymization for road networks[J]. Geoinformatica, 2011, 15(3): 571-607.
- [8] CHOW C Y, MOKBEL M F, LEONG H V. On efficient and scalable support of continuous queries in mobile peer-to-peer environments[J]. IEEE Transactions on Mobile Computing, 2011, 10(10): 1473-1487.
- [9] AMBAINIS A. Upper bound on the communication complexity of private information retrieval[M]. Berlin: Springer, 1997: 401-407.
- [10] KUSHILEVITZ E, OSTROVSKY R. Replication is not needed: single database, computationally-private information retrieval[C]//Annual Symposium on Foundations of Computer Science. c1997: 364-373.
- [11] GHINITA G, KALNIS P, KHOSHGOZARAN A, et al. Private queries in location based services: anonymizers are not necessary[C]//The ACM Sigmod International Conference on Management of Data. c2008:121-132.
- [12] FUNG E, KELLARIS G, PAPADIAS D. Combining differential privacy and pir for efficient strong location privacy[M]//Advances in Spatial and Temporal Databases. Berlin: Springer, 2015: 295-312.
- [13] YU M, YANG K, WEI L, et al. Practical private information retrieval supporting keyword search in the cloud[C]//2014 6th International Conference on Wireless Communications and Signal Processing (WCSP). c2014:1-6.
- [14] 张峰, 倪巍伟. 基于伪随机数加密的保护位置隐私近邻查询方法[J]. 华东师范大学学报(自然科学版), 2015(5):128-142.
ZHANG F, NI W W. Protect the location privacy neighbor query methods based on the pseudo random number of encryption [J]. Journal of East China Normal University(Natural Science Edition), 2015(5): 128-142.
- [15] SHVACHKO K, KUANG H, RADIA S, et al. The Hadoop distributed file system[C]//The 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE. c2010:1-10.
- [16] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]//Usenix Conference on Networked Systems Design and Implementation (NSDI).c2012:141-146.
- [17] PAPADOPOULOS S, BAKIRAS S, PAPADIAS D. pCloud: a distributed system for practical PIR[J]. IEEE Transactions on Dependable and Secure Computing, 2012, 9(1): 115-127.
- [18] MAYBERRY T, BLASS E O, CHAN A H. PIRMAP: efficient private information retrieval for MapReduce[M]//Financial Cryptography and Data Security. Berlin: Springer, 2013: 371-385.
- [19] BLASS E O, DI P R, MOLVA R, et al. PRISM-privacy-preserving search in MapReduce[M]//Privacy Enhancing Technologies. Berlin: Springer, 2012: 180-200.
- [20] 刘金义, 刘爽. Voronoi 图应用综述[J]. 工程图学学报, 2004,

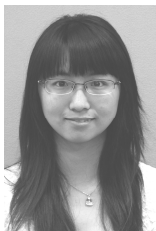
25(2): 125-132.

LIU J Y, LIU S. Voronoi diagram application review[J]. Journal of Engineering Graphics, 2004, 25(2): 125-132.

[21] ASC sequoia benchmark codes[EB/OL]. <https://asc.llnl.gov/sequoia/benchmarks/>.

[22] GMP[EB/OL]. <http://gmplib.org/>.

作者简介:



邓诗卓 (1990-), 女, 辽宁凌源人, 东北大学博士生, 主要研究方向为大数据、云计算、位置隐私保护。



陈月梅 (1991-), 女, 湖北大冶人, 东北大学硕士生, 主要研究方向为隐私保护、大数据、数据管理。



袁野 (1981-), 男, 辽宁沈阳人, 东北大学教授, 主要研究方向为云计算、大数据管理、P2P 计算。



姚继涛 (1990-), 男, 山东泰安人, 东北大学硕士生, 主要研究方向为大数据、云计算、隐私保护。



李艳辉 (1989-), 女, 黑龙江讷河人, 东北大学博士生, 主要研究方向为查询处理、隐私保护。



王波涛 (1968-), 男, 山东福山人, 东北大学教授、博士生导师, 主要研究方向为隐私保护、大数据、云计算、基于位置服务、数据流管理系统、发布/订阅系统、移动数据库。



王国仁 (1966-), 男, 湖北黄冈人, 东北大学教授、博士生导师, 主要研究方向为 XML 数据管理、查询处理优化、高维数据索引、分布式数据库、隐私保护。