一种基于 Hadoop 平台的新聚类算法

缪裕青[1,2] 张锦杏[1] 刘少兵[1] 文益民[1] 明 媚[1]
(桂林电子科技大学计算机科学与工程学院 桂林 541004)[1]
(桂林电子科技大学广西可信软件重点实验室 桂林 541004)[2]

摘 要 针对现有很多聚类算法不能有效处理大规模数据的问题,基于微簇和等价连接关系,提出一种能在 Hadoop 平台实现高效并行化的聚类算法 bigKClustering。算法将紧凑的数据抽象成一个向量,然后通过等价关系对这些向量进行连接,得到最终的聚类结果。实验结果表明,bigKClustering 算法不仅具有良好的时间效率和聚类效果,而且具有良好的可伸缩性、加速比和时间稳定性。

关键词 微簇,等价连接,Hadoop 平台,聚类
中图法分类号 TP311.13 **文献标识码** A

New Clustering Algorithm Bases on Hadoop

MIAO Yu-qing[1,2] ZHANG Jin-xing[1] LIU Shao-bing[1] WEN Yi-min[1] MING Mei[1]
(School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)[1]
(Guangxi Key Laboratory of Trustworthy Software, Guilin University of Electronic Technology, Guilin 541004, China)[2]

Abstract Hadoop is a popular platform to handle huge datasets. But many clustering algorithms can not run effectively over it, for it lacks built-in support for iterative programs, which arises naturally in many clustering applications. We proposed bigClustering which can be easily parallelized in Hadoop MapReduce and done in quite a few MapReduce rounds. Our algorithm is based on the ideas of micro-cluster and equivalence relation. It divides a dataset into many groups and constructs one micro-cluster, which will be treated as a single point, corresponding to each group. All micro-clusters that are closed enough will be connected and put into the same group by the equivalence relation. The center of each group will be calculated and will be the center of a real cluster in the dataset. Experiments show that bigKClustering not only runs fast and obtains high clustering quality but also scales well.

Keywords Micro-cluster, Equivalence relation, Hadoop, Clustering

1 引言

随着数据规模的急速膨胀,对大规模数据进行聚类分析已成为一种现实需要和必然趋势。一个现实的案例是:对社交网络中的用户进行聚类,即按照某些关联标准将关系比较紧密的用户划分到同一个社区。这样做有助于更好地预测一个用户群的消费习惯,更有针对性地制定营销方案。然而,该案例涉及到的网络用户关系图的规模是非常大的,可能包含着数万亿条边,其中每条边代表两个用户间的关系[1]。这种情况下,单一的节点难以满足数据存储和处理的要求,现有的很多串行聚类算法都无法使用。

MapReduce[2]是云计算环境下一种高效的分布式计算框架,使用 MapReduce 处理大规模数据有两个优点:1)用户无需关心分布式环境下数据的存储、分布、拷贝和负载均衡等方面的细节问题;2)采用函数式编程理念,用户只需编写一个 map 函数和 reduce 函数就可完成算法的功能。基于这些特点,MapReduce 在大规模数据分析领域中得到了日益广泛的应用。文献[3,4]分别在 MapReduce 上实现了 k-means 算法和 k-center 算法的并行化,但算法每次迭代都需要处理整个数据集,会产生大量的中间数据。同时,过多迭代所造成的总体通信和带宽开销也不容忽视。文献[5]提出了基于采样的聚类方法并在 MapReduce 上实现并行,但该方法在聚类阶段采用 k-means 方法,同样面临着通信和带宽的问题。本文旨在借助 MapReduce 研究大规模数据的聚类,设计一种新的聚类算法,目标是:1)高效性,算法在串行环境下运行应该能获得良好的聚类质量和时间效率;2)可伸缩性,算法应该能够很好地在 MapReduce 框架上实现并行化,并尽量避开 MapReduce 框架本身在处理数据时所具有的局限,减少中间数据的产生和通信、带宽的开销。

本文提出的算法(bigKClustering)借鉴了 k-means 算法

到稿日期:2013-06-03 返修日期:2013-10-26 本文受广西可信软件重点实验室研究课题(KX201116),广西教育厅科研项目(201204LX122)资助。

缪裕青(1966—),女,博士,副教授,主要研究方向为数据挖掘、分布式计算、云计算,E-mail:miaoyuqing@guet.edu.cn;张锦杏(1986—),男,硕士生,主要研究方向为数据挖掘、云计算;刘少兵(1973—),男,硕士,讲师,主要研究方向为云计算、信息安全;文益民(1969—),男,博士,教授,主要研究方向为数据挖掘、机器学习、图像检测;明 媚(1988—),女,硕士生,主要研究方向为数据挖掘。

的思路和微簇[6]的思想，结合了自定义的等价连接关系。首先，生成微簇；然后，将比较近的微簇连接到一起形成微簇团；最后，求出微簇团的中心。该中心就是聚类中心。实验结果表明，bigKClustering 算法有如下优点：1）串行时聚类质量高，速度快；2）易于在 MapReduce 模型上实现并行，只需少数几个 Job 就能完成，不需进行类似 k-means 算法的迭代操作；3）并行时，算法不仅保持了串行时的性能，且具有很高的时间稳定性和良好的伸缩性及加速比。

2 相关工作

微簇是数据流聚类和大数据集聚类中常用的一种数据总结技术。它是聚簇特征向量[7]的一种拓展，能够有效地保存当前聚簇的数据特征。也就是说，可以用一个微簇来代表一系列数据及这些数据的主要分布特征。根据文献[6,7]，聚簇特征向量和微簇定义如下。

定义 1 聚簇特征可以表示成一个元组$\overrightarrow{CF}=(N,\overrightarrow{LS},\overrightarrow{SS})$。其中，$N$ 代表簇中数据对象的个数，$\overrightarrow{LS}$表示 N 个数据的线性代数和，$\overrightarrow{SS}$表示 N 个数据的平方和。通过该向量可以计算出簇中数据的平均值和方差。

定义 2 微簇在$\overrightarrow{CF}$中加入了时间戳的归总信息，用于流聚类。一个微簇可以表示成向量$\overrightarrow{MC}=(\overrightarrow{CF2^x},\overrightarrow{CF1^x},CF2^t,CF1^t,n)$。其中，$\overrightarrow{CF2^x}$、$\overrightarrow{CF1^x}$、$n$ 分别对应$\overrightarrow{CF}$中的$\overrightarrow{SS}$、$\overrightarrow{LS}$、N，$CF2^t$ 和 $CF1^t$ 分别表示时间戳 $T_{i_1}\cdots T_{i_n}$ 的平方和和代数和。

Aggarwal 等[6]对整个流数据的聚类过程表示如下：首先，在本地生成若干数量微簇。只要内存允许，微簇数量可以尽量多。然后，在线更新微簇。对每个新到来的数据，判断是否能将它纳入到已有微簇中。如果能，就更新已有微簇；否则，新建一个微簇。同时，在每个时间戳到达时，将更新后的微簇存储到本地磁盘。最后，将一个微簇当作一个虚拟点，并使用修改后的 k-means 算法对这些点进行聚类。

MapReduce 最早由 Google 公司提出并实现，Hadoop 平台集成了该框架并被 Yahoo!、Facebook 等公司用于大规模数据的分析。很多算法（如字数统计、链表转置等）都能很自然地迁移到 MapReduce 框架上。但是，对于需要迭代计算的数据分析应用（如聚类、社交网络分析等），MapReduce 并没提供很好的支持。用户要实现这种功能，就必须手工设计多个 MapReduce job，并通过 driver 程序协调这些 job 的执行。然而，这种做法会产生两个问题：1）每一轮迭代都需要重新加载和重新处理数据，即使所有迭代操作所涉及的绝大部分数据可能是不变的。这就浪费了 I/O、网络带宽和 CPU 资源。2）为了判断迭代的收敛状态，在每一次迭代操作后可能需要一个额外的 job 去处理。这又增加了任务调度、数据读取和传输的负担。

Ekanayake 等[8]提出一个基于流的轻量级 MapReduce 框架 Twister，为迭代计算提供了良好支持。它首先将数据分为静态数据（每轮迭代都不变的数据）和动态数据（每轮迭代操作的处理结果），这些数据被存放在分布式内存中；然后采用可缓存的 mappers/reducers 机制将 map/reduce 任务和静态数据统一配置，避免在每次迭代时重新加载静态数据；同时，它也提供了一个“combine”操作，允许用户对所有 reduce 阶段的输出进行汇总，进而直接判断收敛条件。但 Twister 的流架构对节点失效敏感，缺乏有效的容错机制。另外，Twister 将所有数据放在分布式内存中，不适合单节点内存和资源受限的商业机器集群，缺乏良好的伸缩性。

Zhao 等[3]在 Hadoop 平台上实现 k-means 的并行化。在 map 操作中求出数据点到最近聚类中心的距离，在 reduce 操作中更新聚类中心。设计了一个 combine 操作，减少了迭代过程中所产生的中间值，降低了通信开销。Bahman 等[5]通过概率采样的方式为 k-means 选择初始聚类中心，然后使用文献[3]的做法完成聚类。Alina 等[4]提出 k-center 和 k-median 问题的近似算法即 MapReduce-kCenter 算法和 MapReduce-kMedian 算法。两个算法首先使用迭代技术对原始数据集进行采样，得到一个具有良好全局代表性的数据子集，然后在子集上运行 k-center 算法和 k-median 算法。这类算法解决了大数据量的处理问题，但是 MapReduce 在执行这些算法的迭代操作时会产生大量的 I/O 操作和通信负担。

3 bigKClustering 算法

3.1 相关定义

· 微簇

本文将微簇定义为一个$(3d+2)$维的向量$(n_i,\overrightarrow{CF1_i^x},\overrightarrow{CF2_i^x},\overrightarrow{Center_i^x},\max_i)$。其中，$d$ 表示原数据向量的维数，i 是聚簇标识（也是微簇标识），n_i 表示聚簇成员的个数，$\overrightarrow{CF1_i^x}$、$\overrightarrow{CF2_i^x}$分别表示聚簇成员第 x 维的线性代数和与平方和，$\overrightarrow{Center_i^x}$表示聚簇成员第 x 维的中心（均值），$\max_i$ 表示聚簇中所有成员到聚簇中心的最大距离。

· 等价关系

对于任意两个微簇 S_i、S_j，如果 S_i、S_j 的距离不大于一个给定的连接阈值 d，或存在一串微簇$(S_i,S_{i1},S_{i2},\cdots,S_j)$，其中相邻微簇的距离都不大于 d，那么 S_i、S_j 就是具有等价关系的两个微簇。两个微簇 S_i、S_j 的连接距离$dis(S_i,S_j)=dis(\overrightarrow{Center_i},\overrightarrow{Center_j})-\max_i-\max_j$。其中，距离的计算使用欧几里德公式。如果距离为负数，则取 0。距离为 0 时，如果$\overrightarrow{Center_i}$与$\overrightarrow{Center_j}$的距离小于 $\max_i$ 或 $\max_j$，则 S_i、S_j 也具有等价关系。具有等价关系的所有微簇将被连接到一起，并标记为一个微簇团。

3.2 算法思想

已有的大多数聚类算法都使用迭代的方法优化聚类结果。在每次迭代的过程中，原始数据集都会被重新划分。也就是说，在这个过程中，那些本来就比较紧凑的数据可能会被拆散并归并到不同的聚簇中，从而降低了优化的效果。bigKClustering 算法首先将数据集中比较紧凑的数据对象划分到一起，构造出一系列的微簇。这些微簇被当作一个向量来操作，可以保证本来就比较紧凑的数据对象始终属于同一个聚簇。微簇的个数（bigK）大于实际的聚簇数目，但远小于数据向量的个数。只要内存允许，微簇的数量可以尽量大。

得到微簇之后，bigKClustering 算法将使用等价关系将比较邻近的微簇连接到一起，形成一个微簇团。根据微簇所保留的信息，可以求出一个微簇团的中心。该中心就是一个最终聚簇的中心。在调节等价连接的距离时，选择所有微簇中 max 的中值 m 作为初始值。

3.3 bigKClustering 算法描述

算法伪代码描述如下：

算法 1 bigKClustering (ds,bigK,k)

输入：ds（数据集）；bigK（微簇个数）；k（最终聚簇个数）

输出：最终聚类结果

步骤：

1. 从 ds 中任意选择 bigK 个点作为初始中心点；
2. 将每个数据划分到最近的中心点，形成 bigK 个数据块；
3. 对应于各个数据块，总共构造出 bigK 个微簇 $s_1, s_2, \cdots, s_{bigK}$；
4. 计算等价连接的初始距离 m；
5. 等价连接：joinToGroups($\langle s_1, s_2, \cdots, s_{bigK} \rangle$, k, m)，生成 k 个微簇团；
6. 求出各个微簇团的中心，并把它们当作最终聚簇的中心；
7. 将原始数据划分给最近的聚簇中心，并添加聚簇标号。

等价连接函数通过不断地调整连接距离得到 k 个微簇团，其操作过程如下所示：

算法 2 joinToGroups($\langle s_1, s_2, \cdots, s_{bigK} \rangle$, k, d)

输入：$\langle s_1, s_2, \cdots, s_{bigK} \rangle$(bigK 个微簇)；k(微簇团的个数)；d(连接距离)

输出：k 个微簇团

步骤：

```
1)For i=1 to bigK do
    j=i-1;flag=true;
  1.1)if(j!=0)
      For k=1 to j do
    1.1.1)if(dis(si,sk)==0)
        if (dis(Centeri,Centerj)≤maxi || dis(Centeri,Centerj)≤
        maxj)
          flag=false;break
    1.1.2)elseif (dis(si,sk)≤d)
          flag=false;break
  1.2)if(flag==true)
      给 si 赋一个新的微簇团标号
  1.3)else
      将 si 加入到 sk 所在的微簇团中
2)生成微簇团的个数为 k，执行步骤 4)；生成微簇团的个数不等于 4，
  执行步骤 3)；
3)调节连接距离 d，执行步骤 1)；
4)返回微簇团。
```

3.4 算法在 MapReduce 上的并行化 pbigKClustering

容易发现，bigKClustering 算法能够很好地在 MapReduce 模型上实现并行化。首先，步骤 1 到步骤 3 是一个标准的 mapper 和 reducer 操作，使用 1 个 job 完成。在 mapper 端随机选择 bigK 个中心点并将每个向量及其最近的中心点作为一个键/值(中心点作为键)对输出。在 reducer 端分别对各个键及其对应的值进行处理，构造微簇。将所有 reducer 产生的微簇存放到一个文件中；然后，用一个 job 来完成步骤 4 和步骤 5。在 mapper 端读出微簇，计算初始连接距离 m，将这些信息输出。由于微簇的数量相当小，用一个 reducer 就能很快地完成等价连接和聚簇中心的计算，类似于串行算法的做法；最后，用一个 job 完成步骤 6、7。这一步的 mapper 操作与步骤 2 一致，在 reducer 端给每个向量添上簇标号并输出整个聚类算法的结果。算法伪代码如下：

算法 3 pbigKClustering (ds, bigK, k)

输入：ds(数据集)；bigK(微簇个数)；k(最终聚簇个数)

输出：最终聚类结果

步骤：

1. 随机采样 bigK 个点作为初始中心点 $c_{i(i=1,\cdots,bigK)}$；
2. 每个 mapper 负责一个数据分片 $X' \subseteq ds$，并将每个数据 $x_i \in X'$ 划分到最近的中心点 c_i，输出键/值对$\langle c_i, x_i \rangle$；
3. 各个 reducer 对接收到的每个输入$\langle c_i, [x_1, \cdots, x_n] \rangle$构造一个微簇，这样，所有的 reducer 将总共构造 bigK 个微簇 $s_1, s_2, \cdots, s_{bigk}$；
4. 在 mapper 端计算等价连接初始距离 m；
5. 在 reducer 端进行等价连接：joinToGroups($\langle s_1, s_2, \cdots, s_{bigk} \rangle$, k, m)，生成 k 个微簇团并求出各个微簇团的中心 $C_{i(i=1,\cdots,k)}$，C_i 就是最终的聚簇中心；
6. 如步骤 2，将原始数据划分给最近的聚簇中心 C_i；
7. 各个 reducer 为每个键所对应的所有值赋上同一聚簇标号，输出最终聚类结果。

由上可知，我们的并行算法有效地避开了类似 k-means 算法在 job 范围内的迭代操作，大量减少了 MapReduce 执行过程中所产生的中间值和网络流量。同时，我们的并行算法的运行时间主要损耗在微簇的构造上，而这一步是完全并行的操作。所以，我们的并行算法能获得很好的时间效率。

4 实验评估

本文串行算法的实验环境为 Intel(R)Core(TM)i3-2100 CPU 3.1GHz，双核，4.00GB 内存，1TB 外存，Windows 7 操作系统。并行算法在 Hadoop1.0.0 集群上运行。集群由 3 台计算机搭建而成，每台计算机的硬件环境与串行算法运行的环境一致。

4.1 数据集

对于串行算法，我们使用一个真实数据集 Cloud[9] 进行测试。分别对算法的聚类质量和运行时间进行评估。对于并行算法，为了测试算法的有效性，我们生成了一组具有 8 个真实聚簇的语义数据集，每个数据集包含 10000 条记录，对应的维数分别为 2、4、8，每一维的数据都服从正态分布并标准化在[0,1]区间内。数据的生成工具及数据格式文件可从文献[10]获取。为了测试算法的伸缩性和加速比，生成 3 个 10 维的数据集，对应的记录数分别为 $1*10^8$(约 1GB)、$2*10^8$、$3*10^8$。

4.2 实验结果及分析

为了更加客观地评估算法的性能，使用 k-means++[11] 和 k-meansII[7] 中采用的聚类开销(cost)作为聚类质量的评估标准，并分别把这两个算法作为本文串行算法和并行算法的比较对象。同时，选择知名度最高的 k-means 算法作为比较对象。所有实验结果的值取 10 次实验结果的平均值。串行实验结果如图 1、图 2 所示，并行实验结果如图 3—图 6 所示。

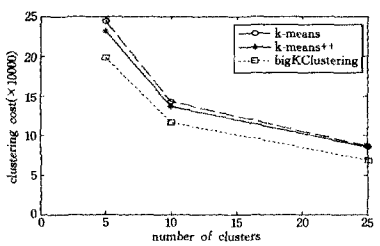

图 1 串行算法的聚类开销

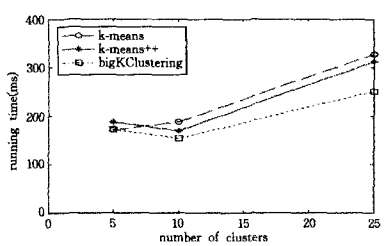

图 2 串行算法的时间开销

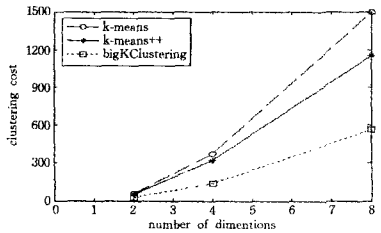

图 3　并行算法的聚类开销

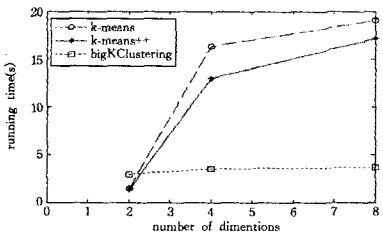

图 4　并行算法的时间开销

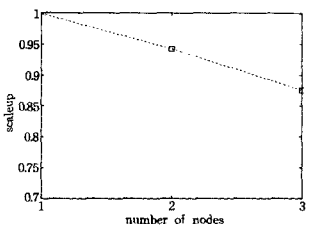

图 5　伸缩性

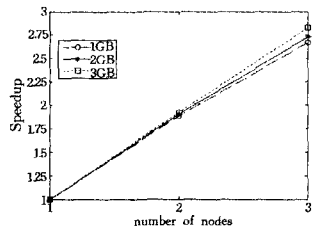

图 6　加速比

实验结果表明，我们的算法无论在串行条件下还是并行条件下，其效率和质量都比参照的算法要好。在并行条件下，我们的算法同时拥有良好的加速比和伸缩性。在图 4 中，数据维数等于 2 时，我们的算法所用的时间比参照算法多。原因主要是：我们的算法的运行时间主要用在微簇的构造上，而构造微簇的时间主要与微簇的数量呈正相关关系；对于另两个算法，它们的运行时间主要由迭代的时间确定，而每次迭代的时间与数据维数呈正相关关系。这也很好地解释了我们的算法的运行时间一直都比较稳定，而参照算法的运行时间随着数据维数的变化，波动比较大。另外，我们也发现：多数情况下，k-means++算法收敛所需迭代的次数确实比 k-means 少很多。在 cloud 数据集下生成 10 个聚簇，运行 10 次的统计结果如表 1 所列。

表 1　每次实验中算法需要的迭代次数（运行 10 次）

	1	2	3	4	5	6	7	8	9	10
k-means	40	42	39	17	46	36	20	42	15	16
k-means++	12	8	20	19	11	14	13	12	23	12
bigKClustering	4	3	6	4	6	5	5	4	4	4

4.3　参数讨论

从理论上来说，微簇个数 bigK 越大，聚类的质量越高。在本文实验中，我们发现在 bigK 取 60 的时候，就能在比较短的时间内获得比较好的聚类结果。对于等价连接距离 d，我们首先初始化为 m，然后使用循环试探的方式对它进行调整。比如，等价连接进行一次后，发现 d 太小（即产生微簇团的数目大于 k），就令 $d=2*m$；如果接着发现 d 太大，就令 $d=1.5*m$。依次类推，直到等价连接产生 k 个微簇团。在实验过程中，我们发现 d 调整的次数很少。在 Cloud 数据集进行 10 次实验，每次生成 10 个聚簇，每次实验所需的调节次数如表 1 所列。对于 k-meansII 算法的参数设置，我们的做法与文献[7]近似：进行 5 次迭代，每次迭代选取 $10*k$ 个点向量，使用 k-means 算法对这些点进行聚类，生成 k 个初始聚类中心。更加详细的说明，请参考文献[7]。

结束语　云计算技术的兴起和发展，为数据挖掘的研究和应用提供了新的机遇。本文借鉴微簇的思想，使用自定义的等价连接方式设计了一种新的聚类算法。算法简单、高效且聚类质量高。在 MapReduce 模型上实现并行化，有效避开了大多同类算法的多迭代操作，取得了很好的时间性能。同时，并行的算法具有良好的时间稳定性、伸缩性和加速比。未来的工作：尝试找出一个有效的表达式来计算等价连接距离，表达式的自变量应是微簇的某些数据信息。这样，在聚类之前就不需要先给定具体的聚簇个数，同时还能省去等价连接的迭代操作。

参 考 文 献

[1] Malewicz G, Austern M H, et al. Pregel: a system for large-scale graph processing[C]// Proceedings of the 2010 international conference on Management of data. Indiana, USA, 2010: 135-146

[2] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Proceedings of Operating Systems Design and Implementation. San Francisco, CA, 2004: 137-150

[3] 赵卫中，马慧芳，傅燕翔，等. 基于云计算平台 Hadoop 的并 k-means 聚类算法设计研究[J]. 计算机科学，2011，38(10)：166-169

[4] Ene A, Im Sung-jin, et al. Fast clustering using MapReduce[C]// Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. California, USA, 2011: 681-689

[5] Bahman B, Benjamin M, et al. Scalable k-means++[J]. Proceedings of the VLDB Endowment, 2012, 5(7): 622-633

[6] Aggarwal C C, Han Jia-wei, et al. A Framework for Clustering Evolving Data Streams [C]// Proceedings of the International Conference on Very Large Data Bases. Berlin, Germany, 2003: 852-863

[7] Zhang Tian, Ramakrishnan R, et al. BIRCH: an efficient data clustering method for very large databases[C]// Proceedings of the 1996 ACM SIGMOD international conference on Management of data. Montreal, Quebec, Canada, 1996: 103-114

[8] Ekanayake J, Li Hui, et al. Twister: a runtime for iterative MapReduce[C]// Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. Chicago, Illinois, 2010: 810-818

[9] http://archive.ics.uci.edu/ml/datasets/Cloud

[10] http://elki.dbs.ifi.lmu.de/wiki/DataSetGenerator

[11] Arthur D, Vassilvitskii S. k-means++: The advantages of careful seeding[C]// Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. New Orleans, Louisiana, 2007: 1027-1035