SCoS：基于Spark的并行谱聚类算法设计与实现

朱光辉 黄圣彬 袁春风 黄宜华

（南京大学计算机软件新技术国家重点实验室 南京 210046）

（江苏省软件新技术与产业化协同创新中心 南京 210046）

摘 要 谱聚类是一种比传统聚类算法更为高效的算法，其建立在谱图理论基础上，并将聚类问题转化为图的最优划分问题.与传统 k-means 算法不同的是，谱聚类算法不仅能够在任意形状的样本空间上实现聚类，而且可以收敛至全局最优解.然而，谱聚类算法的计算开销较大，不仅需要计算任意两个样本之间的相似性，而且还需要计算 Laplacian 矩阵的特征向量.因此，在大规模数据场景下，谱聚类算法存在计算耗时过长甚至无法完成计算的问题.为了解决谱聚类算法在大规模数据场景下的计算性能问题，使得谱聚类算法能够应用在大数据集上，文中基于 Apache Spark 分布式并行计算框架研究并实现了大规模并行谱聚类算法 SCoS，对算法流程中的每个计算步骤进行了并行化.具体的，SCoS 主要实现了相似度矩阵构建与稀疏化过程的并行化、Laplacian 矩阵构建与正规化过程的并行化、正规化 Laplacian 矩阵特征向量计算的并行化以及 k-means 聚类的并行化.为了降低谱聚类算法中大规模样本相似性计算的开销，SCoS 采用了基于多轮迭代的并行计算方式实现大规模样本之间的相似性计算.针对大规模谱聚类算法中耗时较长的 Laplacian 矩阵特征向量求解问题，SCoS 基于 ScaLAPACK 实现了特征向量的并行化求解，同时文中也实现了近似特征向量计算算法，并且对比分析了精确特征向量计算与近似特征向量计算对于谱聚类算法的性能影响.为了进一步提升大规模谱聚类算法的性能，SCoS 采取了矩阵稀疏化表示与存储、Laplacian 矩阵乘法优化以及 k-means 聚类中距离计算放缩剪枝等多种优化手段，尽可能地减少计算开销、存储空间开销以及数据传输开销.实验表明，SCoS 不仅在聚类效果上要优于传统的聚类算法，而且具有较高的运行效率，特别是在大规模数据集下，仍具有较高的计算性能，并表现出了良好的数据可扩展性和系统可扩展性.

关键词 谱聚类；并行化；相似性度量；分布式计算；Apache Spark

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2018.00868

SCoS: The Design and Implementation of Parallel Spectral Clustering Algorithm Based on Spark

ZHU Guang-Hui HUANG Sheng-Bin YUAN Chun-Feng HUANG Yi-Hua

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210046)

(*Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing* 210046)

Abstract Spectral clustering is a more efficient algorithm than traditional clustering algorithms. In spectral clustering algorithm, the clustering problem is transformed into optimal graph partition problem according to the spectral graph theory. Compared with k-means algorithm, the difference is that spectral clustering algorithm is not only able to achieve clustering on the sample space of arbitrary shape, but also able to converge to the global optimal solution. However, the large computation overhead is a main problem of spectral clustering algorithm, which requires not only the calculation of pairwise similarities between all samples but also the calculation of eigenvectors

收稿日期：2016-09-02；在线出版日期：2017-04-06.本课题得到企业合作研究项目“大规模软件分析与系统研究”、国家自然科学基金项目(61572250)和江苏省科技支撑计划项目(BE2017155)资助.朱光辉，男，1987年生，博士研究生，中国计算机学会(CCF)学生会员，主要研究方向为大数据并行处理和大规模机器学习等.E-mail：guanghui.zhu@smail.nju.edu.cn.黄圣彬，男，1993年生，硕士，主要研究方向为大数据并行处理技术和图计算.袁春风，女，1963年生，教授，中国计算机学会(CCF)会员，主要研究领域为体系结构与并行计算、多媒体文档处理、Web信息检索与挖掘等.黄宜华(通信作者)，男，1962年生，教授，中国计算机学会(CCF)会员，主要研究领域为大数据处理与云计算技术、体系结构与并行计算.E-mail：yhuang@nju.edu.cn.

of Laplacian matrix. Therefore, in some big data scenarios, the computation time of spectral clustering algorithm is too long to tolerate and the spectral clustering algorithm may not work efficiently. In order to improve the performance of spectral clustering algorithm in big data, a large-scale parallel spectral clustering algorithm named SCoS is studied and realized in this paper with Apache Spark distributed and parallel computing framework. In SCoS, each computational step in spectral clustering algorithm procedure is implemented in parallel. Specifically, the main steps of spectral clustering algorithm, including similarity matrix construction and sparsification, Laplacian matrix construction and normalization, normalized Laplacian matrix eigenvector computation and k-means clustering are all parallelized in SCoS. In order to reduce the computation overhead of large scale similarity measurement in spectral clustering algorithm, SCoS adopts a novel parallel similarity computation approach based on multi-round iteration technique, which guarantees that the similarity of any two samples is calculated only once and the duplicated computation can thus be avoided. Aiming at getting over the time-consuming Laplacian matrix eigenvector computation problem in large-scale spectral clustering algorithm, an efficiently parallel eigenvector computation method is proposed in SCoS. By integrating and invoking the ScaLAPACK eigenvector computation algorithm MRRR in Apache Spark framework, SCoS improves the performance of eigenvector computation greatly. At the same time, an approximated Laplacian matrix eigenvector computation algorithm is also implemented in this paper and the impact on performance of spectral clustering brought by accurate eigenvector computation and approximated eigenvector computation is analyzed too. To further improve the performance of large scale spectral clustering overall, many other optimization methods are implemented in SCoS. Firstly, sparse matrix representation and storage mode is used in SCoS. Secondly, Laplacian matrix is constructed in parallel by replacing the element at the specified position in similarity matrix with the advantage of sparse matrix representation. In addition, the normalized Laplacian matrix is computed by converting the successive matrix multiplication into corresponding row transformation and column transformation of Laplacian matrix. Finally, in the last k-means step of spectral clustering algorithm, the distance computation process is optimized by the means of avoiding the redundant distance calculation between sample point and center point as much as possible. With these techniques mentioned above, the overheads of computation, storage and data transmission are significantly reduced. The experimental results show that SCoS outperforms traditional clustering algorithm in clustering effect. Furthermore, SCoS still has high computation performance in large-scale datasets and achieves better data and system scalability.

Keywords　spectral clustering; parallelization; similarity measurement; distributed computing; Apache Spark

1　引　言

聚类分析是数据挖掘和机器学习领域中一项基础且重要的技术，它基于数据对象的特征对数据集进行聚类，将相似的数据对象聚在一起，从而发现对象间的内在联系，获取隐藏在数据背后的价值规律[1]．谱聚类是一种基于对象两两之间相似性的聚类算法，通过寻找最优的相似性图的分割方式实现聚类[2-3]．谱聚类算法的聚类效果要优于传统的聚类算法（如 k-means），并在图像分割、文本分析以及社团发现等领域得到了广泛的应用[4-8]．

谱聚类算法虽然具有良好的聚类性能，但是存在计算复杂度高以及计算时间过长等问题．谱聚类算法需要计算样本两两之间的相似度、求解 Laplacian 矩阵的特征向量以及 k-means 聚类，其中每个步骤都涉及到大量的计算，计算开销成为了谱聚类算法主要的性能瓶颈．大数据时代下，随着数据规模的不

断膨胀，谱聚类算法的计算性能问题越来越突出，并严重制约了谱聚类算法在大数据集下的应用.大规模数据场景下，如何降低谱聚类算法的计算复杂度已经成为了一个具有挑战性的问题.近年来，随着MPI、MapReduce[9]等并行计算模型以及Hadoop、Spark[10]等诸多大数据分布式并行计算框架出现，聚类分析算法的并行化逐渐得到了广泛的关注和研究.一些研究人员开展了k-means算法的并行化研究工作[11-12]，并取得了良好的效果.除此之外，谱聚类算法的并行化研究工作也到了一定的进展[13-16].

目前，Apache Spark已经成为了主流的大数据处理分析平台，它采用了内存计算模式，兼顾大数据查询分析计算、批处理计算、流式计算以及图计算等多种计算范式.Apache Spark使用了RDD(Resilient Distributed Dataset)的抽象数据结构以及基于DAG(Directed Acyclic Graph)的抽象计算流程，极大提升了复杂机器学习和数据挖掘分析的计算性能.

基于Apache Spark处理大数据的优越性能，本文研究并实现了大规模并行化谱聚类算法SCoS(Spectral Clustering on Spark).为了进一步提升谱聚类算法的性能，SCoS又采用了基于多轮迭代的相似度并行计算、矩阵稀疏化表示与存储、特征向量计算优化、Laplacian矩阵乘法优化以及k-means聚类中距离计算放缩剪枝等多种优化手段.实验表明，SCoS算法不仅具有良好的聚类效果，而且在大规模数据集下表现出了良好的数据和系统可扩展性.

本文的主要贡献可归纳为以下4点：

(1)针对大规模数据集下谱聚类算法存在的计算开销过大以及计算时间过长等存在的问题，研究并设计了并行化谱聚类算法，主要包括相似度矩阵构建与稀疏化过程的并行化、Laplacian矩阵构建与正规化过程的并行化、正规化Laplacian矩阵特征向量计算的并行化以及k-means聚类的并行化.

(2)设计并实现了基于多轮迭代的样本间相似度并行计算算法，保证了每两个样本之间相似度只会计算一次，避免了重复计算的发生.

(3)针对大规模谱聚类算法中耗时较长的Laplacian矩阵特征向量求解问题，基于ScaLAPACK采取了更为高效的精确特征向量并行计算算法MRRR，同时也实现了近似特征向量计算算法，并且对比分析了精确特征向量计算与近似特征向量计算对于谱聚类算法的性能影响.

(4)基于Apache Spark并行计算框架设计实现了大规模并行谱聚类算法SCoS，并针对不同的测试数据集，完成了算法的正确性实验以及扩展性实验.实验表明，SCoS算法不仅具有良好的聚类效果，而且在大规模数据集下表现出了良好的数据和系统可扩展性.

本文第2节详细介绍谱聚类算法的相关工作，包括谱聚类算法的基本原理、主要流程以及谱聚类算法中的相似性度量等；第3节详细介绍谱聚类算法并行化设计过程与优化措施；第4节详细描述SCoS算法的具体实现过程；第5节对SCoS算法的正确性以及可扩展性进行测试和分析；最后对全文进行总结并明确下一步的工作.

2 相关工作

2.1 谱聚类算法

谱聚类算法是一种基于谱图理论的聚类算法，其本质是将聚类问题转化为图的最优划分问题[17].图中的每个节点代表每个样本，节点之间边的权重代表样本之间的相似性，谱聚类算法通过寻找最优的图划分方式，使子图内部边的权重尽可能的大，不同子图之间边的权重尽可能的小.谱聚类算法的聚类效果要优于传统的聚类算法，而且克服了传统聚类算法(如k-means)在非凸样本空间下容易陷入局部最优解的问题.谱聚类算法能够适用于任意的样本空间，且收敛于全局最优解.

图划分方式对于谱聚类算法的性能有着直接的影响，经典谱聚类算法的划分方式主要包括Min Cut[18]、Radio Cut[19]、Min-Max Cut[20]、Normalized Cut[5]，其中应用最为广泛的为Normalized Cut[3].在加权有向图中，Normalized Cut的最优化问题是一个NP-Complete问题，Shi等人采用了一种松弛变换方法，将原问题转换成Laplacian矩阵分解问题.Ng等人给出了基于Laplacian矩阵分解的谱聚类算法的具体流程，首先计算Laplacian矩阵前k个最小特征向量得到降维后的特征向量矩阵，然后再对特征向量矩阵的行向量进行k-means聚类[21].因此，谱聚类算法不受数据样本特征维数的影响，避免了传统的k-means聚类算法由于高维特征向量造成的奇异性以及聚类性能下降的问题.

由于谱聚类算法的诸多优点，近年来，许多研究人员提出了各种改进的谱聚类算法[22-26].另外，还有

一些研究人员将深度学习应用在谱聚类上，通过深度学习来实现图的划分[27-28]．随着流式计算场景的不断增多，Yoo 等人提出了流式谱聚类算法，针对连续的流式数据进行谱聚类分析[29]．谱聚类算法性能优于传统的聚类算法，但是也存在计算复杂高及计算开销大的问题，特别是数据规模较大时，这些问题更加突出．为了提升大规模数据下谱聚类算法的计算性能，一些研究人员通过并行化的方式来提升谱聚类的性能[13-16]，但是文献中的并行谱聚类算法大都基于 MPI(Message Passing Interface)来实现并行计算，不能和当前主流的、已得到广泛使用的大数据生态系统(如 HDFS、Apache Spark 等分布式存储和计算系统)兼容，可扩展性以及容错性较差．另外，这些算法缺少 Laplacian 矩阵构建与正规化过程的并行化实现，只是对相似度矩阵构建、特征向量计算以及 k-means 聚类三个步骤进行了简单的并行化实现，缺少更进一步的并行优化．

考虑到当前并行谱聚类算法中的一些问题，本文基于主流的 Apache Spark 并行计算框架研究并实现了大规模并行化谱聚类算法，把谱聚类算法中相似度矩阵构建与稀疏化、Laplacian 矩阵构建与正规化、正规化 Laplacian 矩阵特征向量计算以及 k-means 聚类 4 个步骤进行了并行化实现和优化．

2.2　谱聚类算法中相似性度量

大多数聚类分析算法都是基于样本之间的距离．由于距离能够刻画样本之间的相似性，使得距离较近的点具有更高的相似性，距离较远的点相似性低．因此，如何选定距离度量的指标，从而更加真实合理地反映样本点之间的近似关系是聚类算法必须考虑和解决的问题．

谱聚类算法中经常使用高斯核函数计算两样本点之间的相似度[30]．高斯核函数定义为：如果样本点 x_i 和 x_j 存在相似性，则两个样本点的相似度为

$$W_{ij}=e^{-\frac{\|xi-xj\|^2}{\sigma}},\ \sigma\in R \tag{1}$$

$\|x_i-x_j\|^2$ 表示两样本点间的欧氏距离，σ 为预先设置的参数．这种度量方式存在的一个明显缺点是参数 σ 值的选取．Zelnik-Manor 等人后来又提出了一种支持参数 σ 自动调整的算法[6]．另外，Meila 等人利用 Markov 链中随机游走的概率定义样本间相似度[31]，Wang 等人提出了基于密度敏感的相似性度量方式[32]，Chang 等人又提出了基于路径的相似性度量方式[33]．

样本之间的相似度矩阵是一个 $N\times N$ 的矩阵(N 为数据样本规模)，如果相似度矩阵是稠密的，则会带来很大的存储空间开销和计算开销，为此，可以考虑将稠密的相似度矩阵稀疏化来降低存储以及计算开销．常用的稀疏化方式有：(1) ε 阈值方式．计算样本之间的相似度，设定阈值 ε，并认为相似度小于该阈值的样本相似度为 0；(2) t 近邻方式．计算每个样本的 t 个最近邻居，并认为该样本与邻居之间存在相似性．但由于近邻关系的非相互性，按此方法构造的相似度矩阵不对称，因此需要将相似度矩阵对称化．一种方法是采取“或”的方式，即如果 $x_i(x_j)$ 是 $x_j(x_i)$ 的 t 近邻点之一，则认为两点存在相似性；另一种方法是采取“与”的方式，如果 x_i 是 x_j 的 t 近邻点之一，且 x_j 也是 x_i 的 t 近邻点之一，则认为两点存在相似性．另外，为了降低相似性度量的计算开销，Fowlkes 等人提出了利用 Nyström 近似算法来避免所有样本之间相似性的计算[34]，通过样本采样的方式只计算部分样本的相似性．Nyström 近似算法能够减少样本相似性度量的计算开销，但是也降低了聚类效果．

为了降低相似性度量的计算开销，本文设计并实现了高效的基于多轮迭代的相似性并行计算算法，并采用了 t 近邻方式实现相似度矩阵的稀疏化．

3　并行化谱聚类算法

为了解决大规模数据集下谱聚类算法的性能问题，本文设计并提出了并行化谱聚类算法，降低算法在计算时间和存储空间上的开销．

3.1　谱聚类算法主要流程

谱聚类算法的基本流程可以归纳为以下 4 个主要步骤：

(1) 相似度矩阵构建及稀疏化．加载数据文件计算样本之间的相似性，构建相似度矩阵 $\boldsymbol{W}$；对相似度矩阵 $\boldsymbol{W}$ 进行稀疏化操作．

(2) Laplacian 矩阵构建及正规化．根据相似度矩阵 $\boldsymbol{W}$ 计算对角矩阵 $\boldsymbol{D}$，矩阵 $\boldsymbol{D}$ 对角线上的值为相似度矩阵 $\boldsymbol{W}$ 中对应行的元素之和，计算公式如式(2)所示．

$$D_{i,i}=\sum_{j}^{N}w_{i,j} \tag{2}$$

计算 Laplacian 矩阵，如式(3)所示．

$$\boldsymbol{L}=\boldsymbol{D}-\boldsymbol{W} \tag{3}$$

正规化 Laplacian 矩阵，如式(4)所示.

$$\boldsymbol{L}=\boldsymbol{D}^{-1/2}\cdot\boldsymbol{L}\cdot\boldsymbol{D}^{-1/2} \tag{4}$$

(3)特征向量计算.计算正规化 Laplacian 矩阵前 k 个最小特征向量，构建特征向量矩阵.

(4)聚类.利用 k-means 算法对特征向量矩阵的每一行进行聚类.

本文将在接下来的部分详细描述各个环节的并行化算法设计方案.

3.2 相似度矩阵构建及稀疏化过程的并行化

相似度矩阵包含了任意两个数据样本之间的相似度信息，而所有样本两两之间相似度的计算过程复杂度为 $O(n^2)$，因此，在大规模数据集下，构建相似度矩阵会带来较大的计算开销.为了提升大规模相似度矩阵构建的性能，本文提出了一种基于多轮迭代的相似度并行计算方法，其复杂度为 $O(n^2/p)$，p 为并发度.另外，根据相似度信息构成的相似度矩阵是一个稠密矩阵，为了降低大规模相似度矩阵的存储空间开销，本文采用 t 近邻方式对原始的相似度矩阵进行稀疏化.

3.2.1 相似度并行计算

在分布式环境下，传统的相似度并行计算方式为：首先把数据分布式存储在各个计算节点，再把所有的数据广播到每个计算节点上，每个节点并行地计算本地数据中的样本与所有数据样本的相似度.这种并行计算方式存在两个问题：(1)存在重复计算，x_i 与 x_j 的相似度会被不同的计算节点计算两次；(2)主节点的压力较大，主节点需要收集汇总各个计算节点的数据.针对以上问题，本文采取了基于多轮迭代的相似度并行计算方法，该方法首先完成了每个数据分区内部样本的相似度计算，然后再完成不同数据分区之间的样本相似度计算，整个计算过程中不需要主节点的参与.在相似度计算时，根据数据样本的序号大小，保证每个样本只和序号比其大的样本进行相似度计算，从而避免了相似度的重复计算.算法示例如图 1 所示，具体流程描述如下：

(1)将数据划分为多个分区存储在不同的计算节点上，假定存在 n 个数据分区，分别为 $p_1,p_2,\cdots,p_n$.为了能够避免重复计算开销，n 一般为奇数；

(2)初始过程.每个分区 p 各自计算分区内部样本之间的相似度，其中，每个样本只和序号比其大的样本进行相似度计算；

(3)多轮迭代过程.在第 k 轮(从 1 开始)迭代

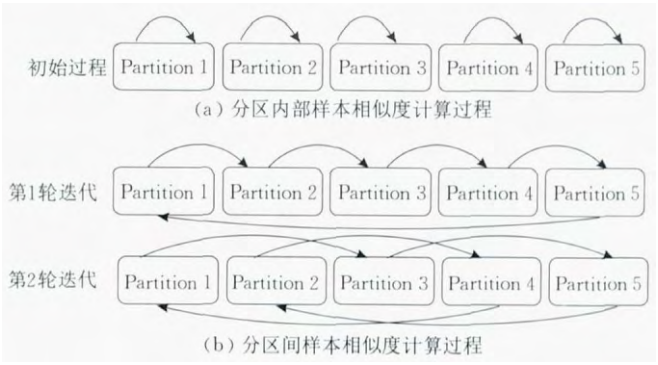

图 1 多轮迭代相似度计算过程

中，分别计算分区 p_i 与 p_j 之间的样本相似度，i,j 为分区标号.其中，$j=(i+k)\%n$，n 是分区个数.总迭次轮数 $m=(n-1)/2$，n 为奇数.当 $k>m$ 时，迭代终止.容易发现，当 n 为奇数，$m=(n-1)/2$ 时可以保证分区间两两不重复、不遗漏地被计算.

3.2.2 相似度矩阵稀疏化

对于大规模的数据集而言，存储稠密的大规模相似度矩阵将会带来很大的存储开销.为此，本文采用 t 近邻的方式对稠密的相似度矩阵进行稀疏化操作.稀疏化会破坏相似度矩阵的对称性，本文又利用倒排索引的思想对 t 近邻稀疏化之后的相似度矩阵进行快速对称化操作.t 近邻的稀疏化方式与设定阈值的方式相比，能够很大程度上确保样本之间相似度信息的完整性，而且实验结果表明，基于 t 近邻的稀疏化方式能够极大地减小存储空间的开销，而且并不会影响谱聚类算法的聚类效果.相似度矩阵稀疏化的处理过程描述如下：

(1)t 近邻过滤.对每个样本 x_i 的相似度信息进行过滤，保留与其距离最近的 t 个邻居，得到过滤后的相似度集合 $SimilaritySet[x_i]$，集合中的数据为 $\{([x_j],sim(x_i,x_j)),\cdots,([x_t],sim(x_i,x_t))\}$，其中 $[x_i]$ 为样本 x_i 的编号，$sim(x_i,x_j)$ 为样本 x_i 和 x_j 的相似度.

(2)对称化.利用倒排索引的思想，以 x_i 邻居的编号作为 key 值，收集并得到 x_i 的每个邻居的相似度集合信息.例如，对于 x_i 的邻居 x_j，通过 $groupBy([x_j])$ 操作，得到倒排后的 x_j 的相似度集合信息 $SimilaritySetReversed[x_j]$，然后将倒排后的相似度集合信息与 $SimilaritySet[x_j]$ 采用“或”的方式进行合并，得到与 x_j 相关的所有相似度信息，进而实现相似度矩阵对称化过程.对称化过程的示例如图 2 所示.

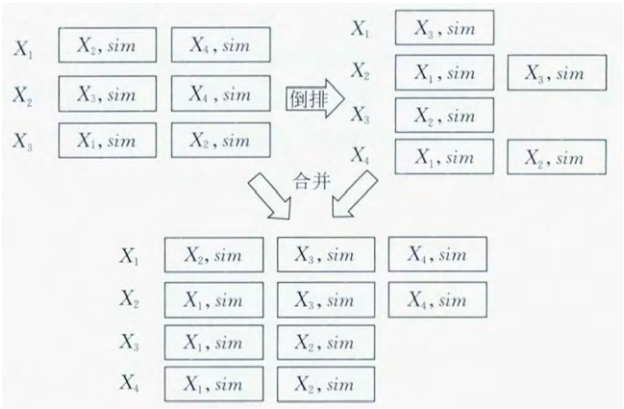

图 2　相似度矩阵对称化过程

3.3　Laplacian 矩阵构建及正规化过程的并行化

3.3.1　Laplacian 矩阵构建

根据式(2)和式(3)所示的矩阵 L 的计算方式，首先需要对相似度矩阵 W 按行求和得到对角矩阵 D，然后将对角矩阵 D 和相似度矩阵 W 相减后得到矩阵 L.对角矩阵 D 的构建可以通过并行地计算每行所有元素的和实现.矩阵 L 可以通过访问两个矩阵同一位置的元素并进行相减得到.矩阵对应位置元素相减的计算方式需要存储两个 $N\times N$ 的矩阵，同时需要 $N\times N$ 次的相减和赋值操作，计算开销较大.通过观察发现，相似度矩阵 W 对角线上的元素均为 0(因为根据相似度的定义，对象与自身的相似度为 0)，而对角矩阵 D 除了对角线上的值之外，其余的值都为 0，另外相似度矩阵 W 的存储采取了稀疏的数据格式，因此本文通过矩阵对应位置元素变换的方式来实现矩阵 L 的构建.并行化步骤如下：

(1)并行构建对角矩阵 D.对稀疏化存储的相似度矩阵 W 以行为计算单位，并行计算每行的所有元素的和，得到对角矩阵 D，如图 3(a)所示.其中，稀疏

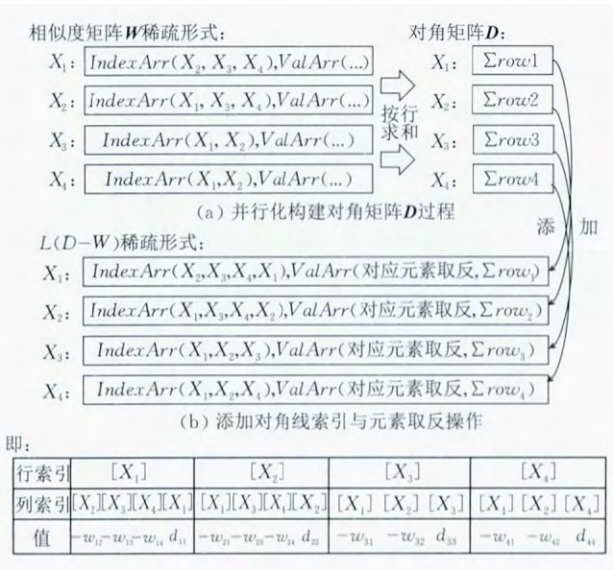

图 3　Laplacian 矩阵并行化构建过程

化存储的矩阵 W 每一行代表了每个样本的相似度集合信息，其中 $IndexArr$ 表示与样本具有相似性的样本编号集合，$VarArr$ 表示对应的相似度值.

(2)W 添加对角索引.对相似度矩阵 W 的每一行添加一个对角线索引标号，且将对角矩阵 D 对应位置的元素值填充到相似度矩阵 W 的对角线位置，如图 3(b)所示.

(3)W 取反操作.对相似度矩阵 W 每行的非对角线位置上的元素值取反，如图 3(b)所示.

通过上述步骤之后，得到 Laplacian 矩阵 L，矩阵 L 的稀疏存储形式如图 3 所示.

3.3.2　Laplacian 矩阵正规化

根据式(4)所示，正规化 Laplacian 矩阵是通过矩阵连乘操作实现，计算复杂度为 $O(n^3)$.因此，本文对 Laplacian 矩阵乘法操作进行了优化.由于 D 是一个对角矩阵，根据对角矩阵性质，对其求幂就相当于对对角线上的元素逐个进行求幂，结果仍是一个对角矩阵.当一个对角矩阵右乘上一个矩阵的时候，等同于将对角线上的元素与矩阵对应行上的元素逐个相乘，相当于对矩阵进行了一次行变换；当一个对角矩阵左乘上一个矩阵的时候，等同于将对角线上的元素与矩阵对应列上的元素逐个相乘，相当于对矩阵进行了一次列变换.具体过程如图 4 和图 5 所示.

$$\begin{bmatrix} a & & \\ & k & \\ & & c \end{bmatrix}\times\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{k1} & \cdots & x_{kn} \\ x_{n1} & \cdots & x_{nn} \end{bmatrix}=\begin{bmatrix} ax_{11} & \cdots & ax_{1n} \\ kx_{k1} & \cdots & kx_{kn} \\ cx_{n1} & \cdots & cx_{nn} \end{bmatrix}$$

图 4　对角矩阵右乘矩阵示意

$$\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{k1} & \cdots & x_{kn} \\ x_{n1} & \cdots & x_{nn} \end{bmatrix}\times\begin{bmatrix} a & & \\ & k & \\ & & c \end{bmatrix}=\begin{bmatrix} ax_{11} & ..kx_{1k}.. & cx_{1n} \\ ax_{k1} & ..kx_{kk}.. & cx_{kn} \\ ax_{n1} & ..kx_{nk}.. & cx_{nn} \end{bmatrix}$$

图 5　对角矩阵左乘矩阵示意

通过上述优化操作，将矩阵的连乘操作通过标量乘矩阵的方式实现，从而完成了矩阵乘法的并行化过程，计算复杂度由原先的 $O(n^3)$ 降为 $O(n^2/p)$，p 为并发度.具体设计流程描述如下：

首先，矩阵 L 是以行向量的稀疏形式存储在各个计算节点上.考虑到对角矩阵的数据存储空间要远小于矩阵 L，为了减少各个节点之间的通信开销，因此，可以先收集对角矩阵 $D^{-1/2}$ 的对角线元素，存放于数组中，记为 $Arr(D^{-1/2})$，并将其分发到各个计算节点上，计算每个节点获取到对角矩阵 $D^{-1/2}$ 的元素，进行正规化矩阵 L 的过程.

$D^{-1/2}\times L$ 并行化计算过程：各个计算节点根据

存储在本地的矩阵 L 的行向量的行号，从 $Arr(\boldsymbol{D}^{-1/2})$ 中取得行号索引位置的元素，与行向量中的每个元素相乘，从而实现并行化的乘法过程.得到的中间结果记为 L'，$L'=\boldsymbol{D}^{-1/2}\times L$.实现过程如图 6所示.

$L'\times\boldsymbol{D}^{-1/2}$ 并行化计算过程：各个计算节点根据存储在本地的中间结果矩阵 L' 的行向量，通过行向量中每个元素的索引号从 $Arr(\boldsymbol{D}^{-1/2})$ 中获取相同索引位置的元素进行相乘，得到的结果即为正规化 Laplacian 矩阵，记为 $\boldsymbol{L}_{Norm}$.实现过程如图 7 所示.

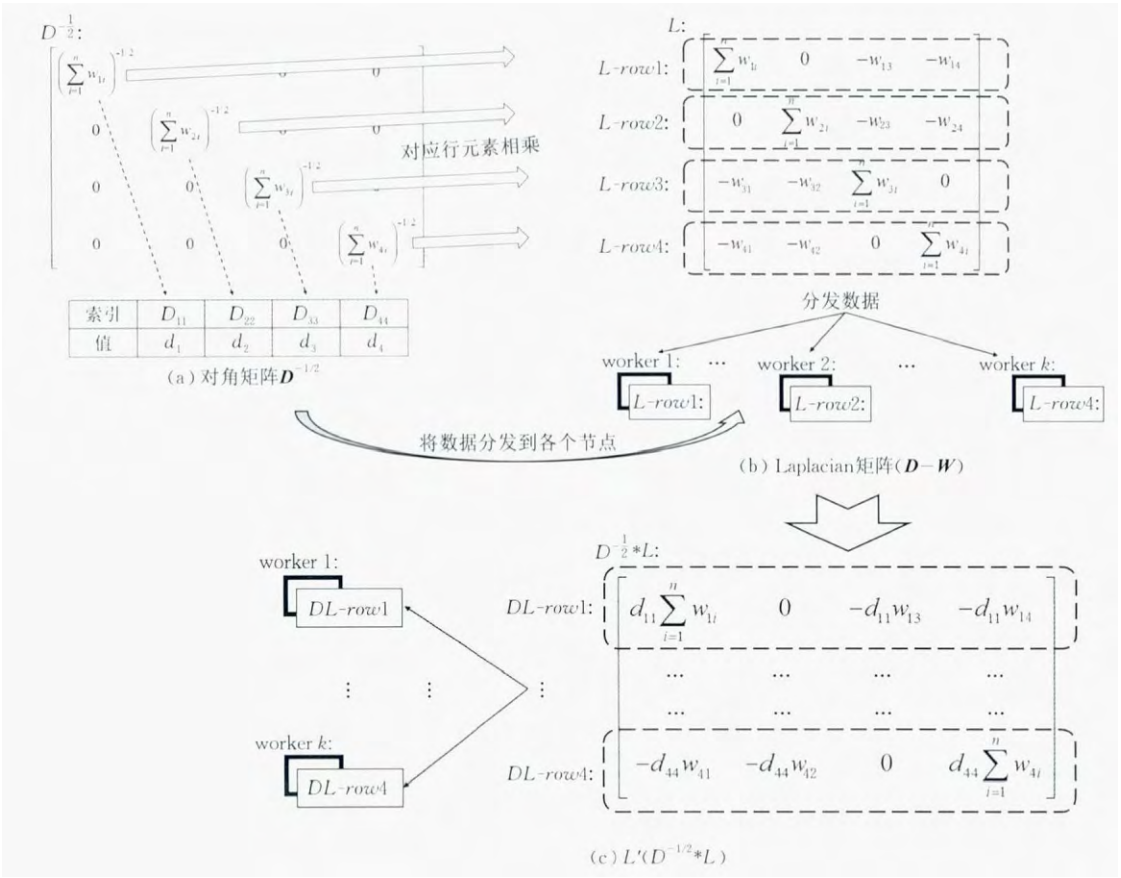

图 6 构建 $\boldsymbol{L}_{Norm}$ 中间过程结果过程示意

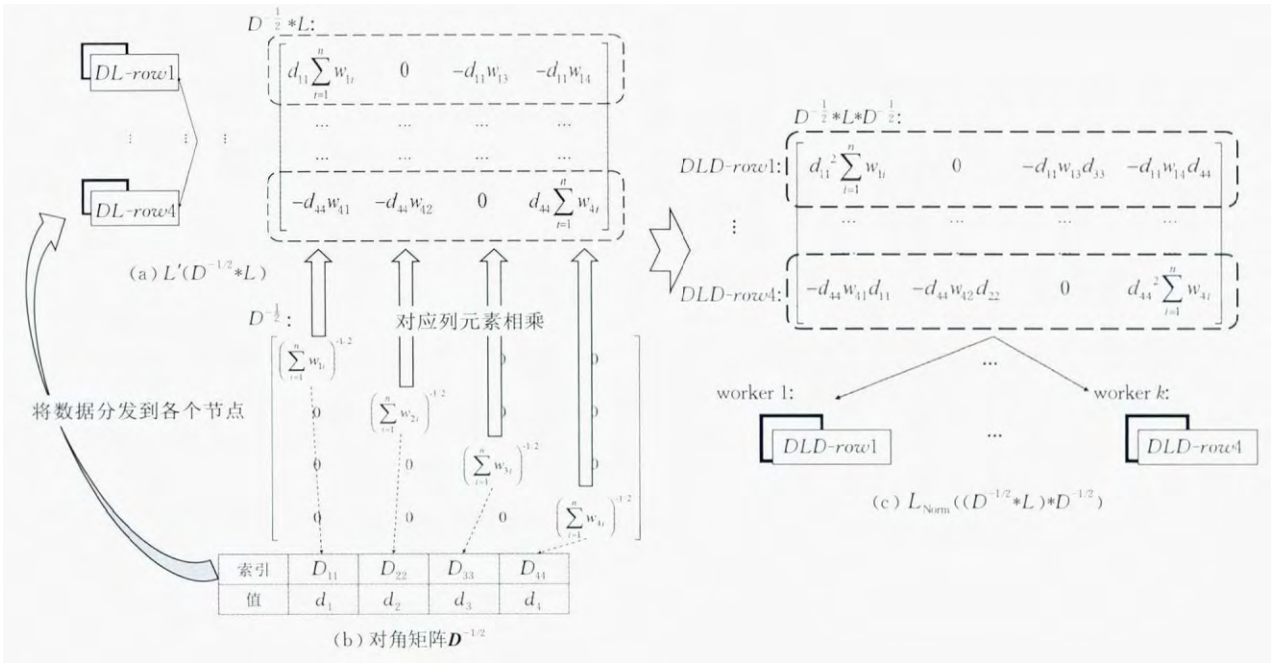

图 7 构建 $\boldsymbol{L}_{Norm}$ 并行化过程示意

3.4　特征向量计算并行化

得到正规化的 Laplacian 矩阵 $\boldsymbol{L}_{Norm}$ 之后，需要求解其特征向量，并选取前 k 个最小的特征值对应的特征向量按列摆放，构成降维之后的矩阵.

目前，大多数主流的线性代数工具包，例如 ARPACK① 和 LAPACK② 等虽都能够支持矩阵的特征值问题求解，但大多是基于单机实现的，针对大规模矩阵时，尤其是大规模特征问题的计算，存在性能瓶颈.目前主流的大数据并行计算框架 Apache Spark 在处理特征值计算时也是通过调用 ARPACK 算法库求解，其主要过程是：各个计算节点将数据发送到主节点，主节点收集所有的部分数据后，构建出整个矩阵并调用 ARPACK 算法库进行单机计算.通过实验发现，当矩阵规模为 $50\,000\times50\,000$ 时，这种单机计算方式的时间开销在两个小时以上.针对目前计算大规模特征问题存在耗时过长的情况，本文设计实现了精确特征向量并行化计算和近似特征向量并行化计算两种方法，旨在加快特征问题求解过程.

3.4.1　精确特征向量求解并行化方法

谱聚类算法中并不需要计算所有的特征向量，只需要求解前 k 个最小特征向量.针对这一特点，本文采取了 MRRR 算法来计算 $\boldsymbol{L}_{Norm}$ 的前 k 个最小特征向量.MRRR 算法计算三对角矩阵前 k 个特征向量的时间复杂度是 $O(kn)$，n 为矩阵的规模[35].为了能够利用 MRRR 算法计算特征向量，本文选用了主流的可扩展的数值线性代数计算库 ScaLAPACK③.ScaLAPACK 是可扩展的 LAPACK，能够实现高效的并行化处理大规模矩阵运算，并且提供了并行化的 MRRR 算法实现[36].并行化的 MRRR 算法的计算复杂度为 $O(kn/p)$，n 为矩阵规模，p 为计算节点个数，具有很好的可扩展性.由于 ScaLAPACK 通过使用 MPI 来完成并行计算，Spark 计算平台并没有提供可以直接调用的 ScaLAPACK 算法库的接口，因此，本文设计并实现了 Spark 平台与 ScaLAPACK 的交互框架，完成了并行化求解特征问题的过程，加快了原先 Spark 平台求解矩阵特征问题的速度.交互框架使用了 pbdR④ 工具包实现对 ScaLAPACK 的调用.整个交互框架如图 8 所示，主要包含以下 3 个部分：

(1) Spark 端.将计算得到的正规化 Laplacian 矩阵结果并行化地写入到 HDFS 中；

(2) Spark R 端.① 由于 MPI 不能够直接识别分布式文件系统 HDFS 中的数据，所以通过 R 读入 Spark 计算得到的结果；② 为了实现并行计算，pbdR 的输入是一个分块矩阵.而 HDFS 存储的矩阵是按行存放的，因此需要通过分布式的 shuffle 过程将 HDFS 中的数据转换为 pbdR 所规定的分块的数据结构.转换结束后，pbdR 在多个计算节点上启动 MPI 并行计算进程.

(3) MPI 端.各个计算节点上的 MPI 程序读取转换后的矩阵数据，调用 ScaLAPACK 库，并行计算矩阵的特征向量.

通过 Spark 与 ScaLAPACK 交互的方式并行化求解特征问题具有良好的可扩展性，虽然存在一定的 I/O 开销，但相比较于 Spark＋ARPACK 单机计算大规模特征问题的求解方法，仍具有很大的优势.

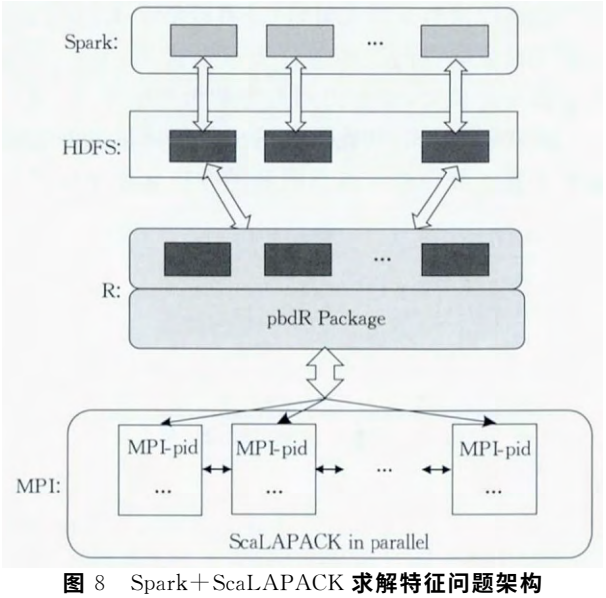

图 8　Spark＋ScaLAPACK 求解特征问题架构

3.4.2　近似特征向量求解并行化方法

除了通过 ScaLAPACK 并行求解特征问题外，本文还提出了一种快速求解矩阵特征向量的方法，其思想来源于 PIC(Power Iteration Clustering)算法[37]，PIC 通过迭代计算的方式近似计算前 k 个最大特征向量.

传统谱聚类算法的思想是求解 Laplacian 矩阵前 k 个最小特征向量，使得在样本空间中存在较高相似度的样本在降维后的空间中尽可能的靠近.谱聚类算法的目标优化函数为

$$\min \sum_{ij} W_{ij} \| y_i - y_j \|^2 \tag{5}$$

从最大可分性出发，将原始最小化问题转换为

① http://www.caam.rice.edu/software/ARPACK/
② http://www.netlib.org/lapack/
③ http://www.netlib.org/scalapack/
④ http://pbdr.org/

如下目标函数的优化问题，从而使投影后样本点的方差最大化.

$$\max -\sum_{ij} W_{ij} \| y_i - y_j \|^2 \tag{6}$$

该目标函数的矩阵向量形式为[30]

$$\max tr(\boldsymbol{Y}^{\mathrm{T}}(-\boldsymbol{L})\boldsymbol{Y}),\ \text{s.t.}\ \boldsymbol{Y}^{T}\boldsymbol{D}\boldsymbol{Y}=\boldsymbol{I} \tag{7}$$

利用拉格朗日乘子法，可得

$$(-\boldsymbol{L})y=\lambda \boldsymbol{D}y \tag{8}$$

其中$(-\boldsymbol{L})$为 $\boldsymbol{L}$ 矩阵的对应位置取反，记为 $\boldsymbol{L}_{Inv}$.因此只需对 $\boldsymbol{L}_{Inv}$ 进行特征分解，求得前 k 个最大特征值对应的特征向量，即是原问题的最优解.

根据 PIC 的思想与迭代收敛性理论，求取 $\boldsymbol{L}_{Inv}$ 前 k 个最大特征向量问题可转化为如下迭代问题，

$$v^{i+1}=\boldsymbol{L}_{Inv}v^{i},\ i=0,1,\cdots \tag{9}$$

当迭代次数 i 增大时，$v(i)$可近似为 $\boldsymbol{L}_{Inv}$ 的前 k 个最大特征向量的线性组合表示形式，而对 $v(i)$进行 k-means 聚类，就可得到最终的聚类结果.

由式(9)可知，迭代求解近似特征向量的并行化过程主要是分布式矩阵与向量的并行化乘法操作过程.由于 $\boldsymbol{L}_{Inv}$ 是一个稀疏矩阵，本文采用图来表示稀疏矩阵，并利用图消息传播模型设计实现了迭代计算近似特征向量的过程.对于稀疏度较高的矩阵而言，图消息传播的方式能够减少计算开销以及数据通信开销.

基于图模型的稀疏矩阵与向量的并行化乘法操作过程如图 9 所示，主要包括以下几个步骤：

(1) 初始化.将稀疏矩阵结构转变为三元组结构，$M(i,j)\rightarrow(i,j,M_{ij})$表示顶点 i 到顶点 j 权值为 M_{ij} 的有向边，图 G_M 如图 9(a)所示.

(2) 向量数据分发.将向量 $\boldsymbol{V}$ 的每个元素分发到图 G_M 中对应的列顶点上，如图 9(b)所示.

(3) 乘法计算.将图 G_M 中边上的权值按边的方向，发送到目标顶点上(列顶点)，实现 $M_{ij}\times v_j$ 的计算过程，如图 9(b)所示.

(4) 收集结果.在图 G_M 中按边的相反方向，将目标顶点(列顶点)上的结果发送到源顶点(行顶点)上，得到最终的结果，如图 9(c)所示.

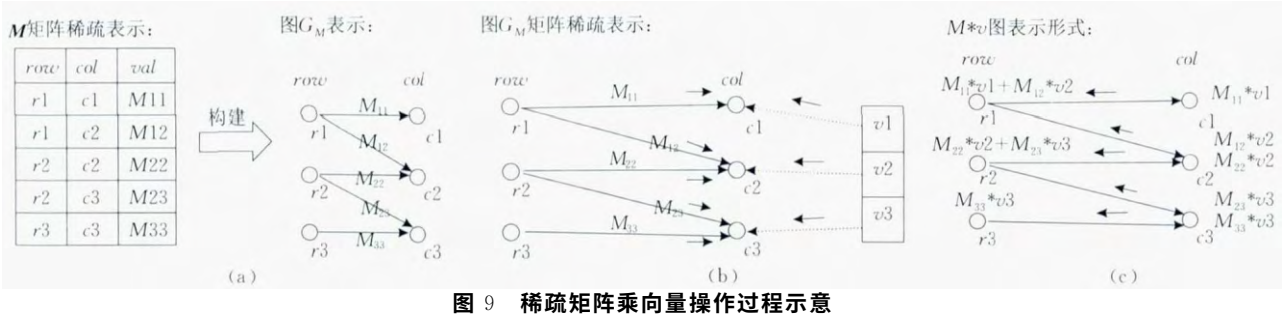

图 9 稀疏矩阵乘向量操作过程示意

3.5 k-means 聚类并行化算法

3.5.1 k-means 聚类算法描述

k-means 算法可将样本对象聚类成 k 个簇(cluster)，具体算法描述如下：

1.利用 k-means++[38] 预采样过程选取 k 个聚类中心点，$P_1,P_2,\cdots,P_k$.

2.对于每一个样本 X_i，计算与其距离最近的中心 P_i，并归属为类 C_i，如式(10)所示.

$$C_i:=\arg\min_j \| X_i - P_j \|^2 \tag{10}$$

3.对于每一个类 j，重新计算该类的中心，如式(11)所示.

$$P_j:=\frac{\sum_{i=1}^{n}\{C_i=j\}X_i}{\sum_{i=1}^{n}\{C_i=j\}} \tag{11}$$

重复步骤 2、3 直到每个样本点到其所属类的中心的距离平方和最小或者当所有样本点的所属类不再发生变化的时候.其中 k 是给定的聚类数，C_i 代表 k 个类中距离样本 X_i 最近的类，即样本 X_i 所属的类为 C_i，所以 C_i 的取值是 1 到 k 中的其中一个.P_j 代表类 j 的样本的中心点的位置.

3.5.2 k-means 聚类并行化过程

并行化 k-means 算法的计算过程的基本流程如图 10 所示，主要分为如下步骤：

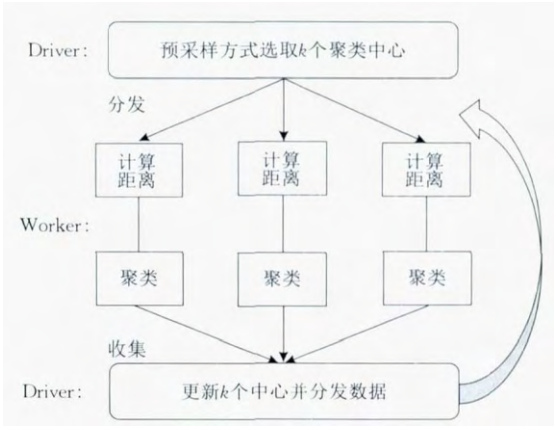

图 10 k-means 计算流程示意

1. 对样本数据预采样得到初始 k 个聚类中心；

2. 将 k 个聚类中心分发到各个处理节点上；

3. 每个处理节点计算本地的样本点到 k 个聚类中心的距离，进行样本聚类；

4. 根据更新后的样本聚类情况，更新 k 个聚类中心；

5. 若算法达到指定迭代次数时，算法停止；否者执行步骤 2.

3.5.3　距离计算优化方法

无论是单机还是分布式的 k-means 聚类算法对于距离的计算通常是直接计算每一个样本到每一个中心的欧氏距离，并选择距离最近的中心，将样本进行归类.这样的方法虽然简单直观，但会造成很大的计算量，尤其是对于分布式算法而言，由于样本数据存放在不同的节点上，更是会带来大量的通信开销.

考虑到计算量和通信量带来的问题，本文采取了将样本数据与其二范数进行关联的优化措施.具体优化过程描述如下：

假设中心点 $Center$ 是 (a_1,b_1)，需要计算距离的点是 (a_2,b_2).首先将数据点的坐标 (x,y) 与其二范数进行关联，构成形如 $\langle(x,y),\|(x,y)\|^2\rangle$ 的键值对形式.则上述两点可表示为：$\langle(a_1,b_1),sqrt(a_1^2+b_1^2)\rangle$ 和 $\langle(a_2,b_2),sqrt(a_2^2+b_2^2)\rangle$ 的 key-$value$ 对形式.并将两点的二范数之差的平方记为 $boundDistance$，如式(12)所示.而真正的欧氏距离(带平方)的值为 $realDistance$，如式(13)所示.

$$boundDistance=\left(\sqrt{a_1^2+b_1^2}-\sqrt{a_2^2+b_2^2}\right)^2 \quad (12)$$

$$realDistance=(a_1-a_2)^2+(b_1-b_2)^2 \quad (13)$$

通过比较两个式子，显然式(12)的结果将会小于等于式(13)的结果.由于 $boundDistance$ 的值是事先就已计算的，且每个样本数据或样本中心都会记录这样一个值.因此在计算距离的时候，比较过程如下：

(1) 先将 $boundDistance$ 的值和当前样本与最近的中心的距离值 $bestDistance$ 进行比较；

(2) 若 $boundDistance$ 都不小于之前计算得到的最小距离 $bestDistance$，则真正的欧氏距离 $realDistance$ 也就不可能小于 $bestDistance$，否则进行步骤(3)；

(3) 当 $boundDistance$ 小于 $bestDistance$ 时，则进行真正的距离的计算.只有当 $realDistance$ 小于 $bestDistance$ 的时候，才更新 $bestDistance$ 的值为 $realDistance$.

上述的优化过程，通过利用样本的 2-范数的差值与样本间欧氏距离的放缩关系，避免了距离的重复计算，降低了 k-means 聚类过程的计算开销.优化流程示意如图 11 所示.

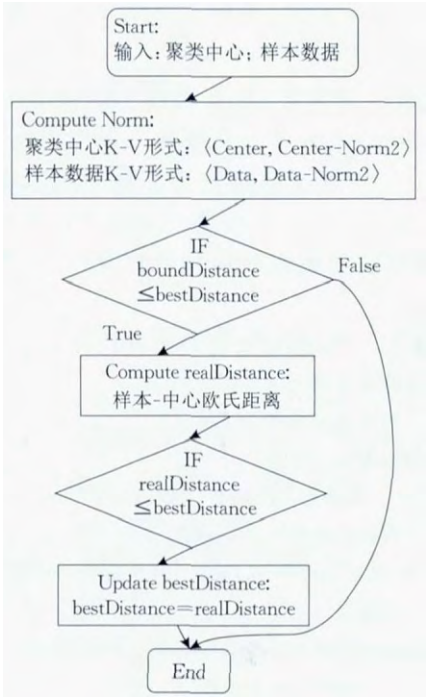

图 11　距离计算优化流程

4　设计与实现

根据上一节描述的算法并行化方案设计，下面就算法实现过程中涉及到的相似度矩阵构建、Laplacian 矩阵正规化、特征向量计算、k-means聚类 4 个部分的实现进行详细介绍.

4.1　相似度矩阵构建

4.1.1　数据样本编号

本算法的输入数据存放于 HDFS 中，每行表示一个数据样本，每个数据样本有若干个特征，特征之间以空格分割.通过 $zipWithIndex$ 算子对数据样本进行编号，编号可作为样本在稀疏矩阵数据结构的位置索引.

4.1.2　相似度并行计算

相似度并行计算主要包括以下几个步骤：

(1) 初始化.通过 $mapPartitionsWithIndex$ 算子实现数据样本分区，并为分区编号；

(2) 初始过程.使用 map 算子实现各个分区内部样本间相似度计算，为避免重复计算，每个样本只需计算与编号大于自身的样本之间相似度；

(3) 迭代过程.通过 map 算子计算每个分区对应的目标分区，分区与目标分区通过 $join$ 算子合并.

4.1.3　构建相似度矩阵

该过程主要包括了数据样本稀疏化和数据样本

对称化两部分操作.

(1)稀疏化操作.采用 map 算子计算每个样本 t 个最近的邻居.

(2)对称化操作.首先通过 $flatMap$ 算子将每个样本的相似度集合信息转为三元组的形式,然后利用 $groupBy$ 算子收集倒排之后的相似度集合信息,再通过 $leftOuterJoin$ 算子将原先的相似度集合信息与倒排后的相似度信息进行合并,实现对称化过程.

算法 1. 构建相似度矩阵并行化算法.

```
输入:数据样本文件
输出:对象间相似度矩阵
buildSimMat(file)
//样本文件数据:file,分区数:blockNum
data←textFile(file,blockNum)
//样本编号:zipWithIndex,分区标号:mapPartitionsWithIndex
data_partition←data.zipWithIndex().
        mapPartitionsWithIndex(pid,data_section){
  pid:Int //表示分区的编号,1,2,…,n
  data_section:Iterator[(Int,Array)] //分区中数据
}
//计算分区内样本间相似度
dis←data_partition.map(computeSimilarity)
//迭代计算分区间样本间相似度
iter←(blockNum-1)/2
FOR (i:iter)
  //目标分区:dst_partition
  dst_partition←data_partition.map((pid+iter)%
      blockNum,data_section).partitionBy(dstPar←
      srcPar%blockNum)
  //分区间相似度计算
  dis←data_partition.join(dst_partition).
       map(computeSimilarity)
END FOR
dis_K:RDD[Int,Array((Int,Double))]←
       dis.map(computeKNN)
dis_tri:RDD[(Int,Int,Double)]←dis_K.flatMap()
//进行倒排
sim_rec:RDD[Int,Array((Int,Double))]←
        dis_tri.map(case(i,j,sim)).groupBy(j)
//合并得到对称相似度矩阵
//SP:〈样本编号,〈index_arr:编号数组,value_arr:相似度数组〉〉
simMat:RDD[SP]←dis_K.leftOuterJoin(sim_rec)
```

4.2 Laplacian 矩阵构建与正规化

计算 Laplacian 矩阵的并行化实现过程主要分为计算 Laplacian 矩阵和正规化两个环节.主要步骤如下:

(1)利用 map 算子求解对角矩阵 $\boldsymbol{D}$,并利用 $Spark$ 的 $broadcast$ 机制将其分发到各个计算节点.

(2)各个节点收到对角矩阵 $\boldsymbol{D}$ 的数据后,通过 map 算子实现矩阵对应位置元素取反并添加对角线索引的操作.

(3)根据对角矩阵的特性,$(\boldsymbol{D}^{-1/2})\times\boldsymbol{L}$ 的过程相当于对角矩阵 $\boldsymbol{D}$ 每行的元素与矩阵 $\boldsymbol{L}$ 对应行的每个元素相乘求得;而 $\boldsymbol{L}\times(\boldsymbol{D}^{-1/2})$ 的过程相当于对角矩阵 $\boldsymbol{D}$ 每行的元素与矩阵 $\boldsymbol{L}$ 对应列的每个元素相乘求得;这两个过程操作可以通过 map 算子进行实现.实现过程如算法 2 所示.

算法 2. Laplacian 矩阵正规化并行化算法.

```
输入:相似度矩阵 W(算法 1 的输出结果)
输出:正规化的 L 矩阵
computeNormLaplacianMatrix(simMat)
diagonalMat←simMat.map(e=>e._2.count).broadcast()
//正规化 Laplacian 矩阵过程
laplacianMat←simMat.map{
FOR each line e
  IF diagonalMat[e]==0
    foreach value in SP's value_arr←0.
  ELSE
    foreach value in SP's value_arr
    value_arr←value_arr*diagonalMat[e]^(-0.5).
  END IF
END FOR
}
laplacianMat←laplacianMat.map{
FOR each line e
  FOR (i:SP.value_arr.length)
    IF diagonalMatr[i]==0
    value_arr[i]←0.
    ELSE
    value_arr[i]←value_arr[i]*
                 diagonalMat[i]^(-0.5).
    END IF
  END FOR
END FOR
}
//laplacianMat 的形式为 RDD[SP]存储形式
laplacianMat:RDD[SP]
```

4.3 特征向量计算

本文对于 $\boldsymbol{L}_{Norm}$ 特征向量的计算采用了实际特征向量和近似特征向量两种求解方式,前者是基于 Spark+ScaLAPACK 的方法,后者利用了图计算模型,通过节点间消息传播机制实现矩阵与向量的乘法.实现过程如算法 3 所示.

算法 3.　近似特征向量求解并行化算法.

```
输入：正规化 Laplacian 矩阵 L_Norm（算法 2 的输出结果）
输出：降维后的结果向量
computePseudoEig(simMat)
//L_Norm 元素求反，得到 L_Inv 矩阵
inverseLMat ←
        laplacianMat.map(doInverse(SP.value_arr))
//构建 L_Norm 的图结构表示形式
edges ← inverseLMat.flatMap(case(i, j, s))
gL ← Graph.fromEdges(edges, 0.0)
v ← gL.aggregateMessage(sendMsg.toSrc,
    mergeMsg.sum).normalize
delta ← Double.MaxValue //设定收敛条件
v_pre ← v  //迭代计算向量 v
WHILE (iter < maxIteration || delta > tol){
  //将向量信息加入到图中，实现向量矩阵乘法
  v_cur ← gL.joinVertices(v).aggregateMessage
          (sendMsg.toSrc, mergeMsg.sum).normalize
  //计算 v^(i) 与 v^(i+1) 之间的误差值
  delta ← v_cur.join(v_pre).map(case(_, x, y) =>
          math.abs(x − y)).sum
  //更新图 gL 与向量 v 的信息
  gL ← GraphImpl.fromExistingRDDs(v_cur, gL.edges)
  v_pre ← v_cur
}
//得到收敛之后的结果即是 L_Inv 的前 k 个最大特征向
  量的线性组合近似
v_cur: VertexRDD[Double]
```

4.4　k-means 聚类

聚类过程采用 k-means 聚类算法.算法的数据输入是由 Laplacian 矩阵降维之后得到的矩阵.首先，设定算法的聚类数 k 与最大迭代次数.利用 k-means++预采样过程选取 k 个数据点作为中心点；并将聚类中心发送到各个节点上，每个节点负责计算本地的样本数据到中心点的距离，并将样本点归类到距离最近的中心所属的类，距离计算的具体实现和优化过程如 3.5.3 部分描述.

算法 4.　k-means 聚类并行化算法实现.

```
输入：Laplacian 矩阵降维之后的结果矩阵
输出：谱聚类算法最终的聚类结果
KMeansCluster(reductionMatrix)
//初始化 k-means 聚类对象，并设置最大迭代次数
  maxIter
Kmeans ← new KMeans().setMaxIterations(maxIter)
//设置聚类中心个数，读取矩阵数据 reductionData
kmeans.setK(clusterNum)
points ← kmeans.loadData(reductionData)
//预采样方式初始化 k 个聚类中心：generateClusterCenter
centers ← kmeans.generateClusterCenter()
result ← kmeans.runAlgorithm{
  WHILE (Iter < maxIter)
  {
    points ← kmeans.computeDistance(centers)
    centers ← kmeans.updateCenters(centers)
  }
}
//聚类结果 result 形式为〈样本编号，样本所属类〉
result: RDD[(Int, Int)]
```

5　实　验

5.1　实验环境与设置

为验证并行化谱聚类算法的性能，本文从算法的算法聚类效果、算法性能、数据可扩展性、节点可扩展性四个方面对算法进行评估.本文采用的实验集群由 1 个主控节点和 8 个计算节点组成，集群的节点配置参见表 1.

表 1　计算节点配置信息

项目	配置说明
CPU	6 Core Intel Xeon 2.10GHz×2
Memory	192GB
Disk	8TB Raid0
Network Bandwidth	1Gbps
OS	RedHat Enterprise Linux Server 7.0
JVM Version	Java 1.7.0
Hadoop Version	Hadoop 2.7.1
Spark Version	Spark 2.0.1
MPI Version	Open MPI 1.8.8

实验所用的数据集包括：(1) 手写数字数据集 MNIST(样本 10000，特征 784，类别数 10)；(2) 新闻语料数据集 20 Newsgroups(样本 18846，类别数 20)；(3) 网络攻击数据集 KDD Cup99(样本约 40 万，特征 41，类别数 23).

为了测试 SCoS 算法的聚类效果，本文选取了 2-NG、20-NG、MNIST 三组数据集作为聚类效果测试数据集，其中，2-NG 数据集由 20 Newsgroups 数据集中标签为 alt.atheism、comp.graphics 对应的样本构成.同时，从 KDD Cup99 数据集中抽取 1000、5000、10000、20000、50000、100000 条样本，并以次命名为 KDD-1、KDD-5、KDD-10、KDD-20、KDD-50、KDD-100 作为 SCoS 算法的扩展性测试数据.所有数据存放于分布式文件系统 HDFS 中.

5.2　聚类效果测试

聚类效果实验是在 MNIST、2-NG、20-NG 这 3 个数据集上进行，并使用 *NMI* (Normalized Mutual Information)作为聚类效果评价标准，*NMI* 公式如

式(14)所示.

$$NMI=\frac{I(\boldsymbol{X},\boldsymbol{Y})}{\sqrt{H(\boldsymbol{X})H(\boldsymbol{Y})}} \tag{14}$$

其中向量 $\boldsymbol{X}$ 为聚类得到的类别序列构成的 n 维向量,向量 $\boldsymbol{Y}$ 表示真实样本标签构成的 n 维向量,n 为样本个数.$I(\boldsymbol{X},\boldsymbol{Y})$为 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 之间的互信息,$H(\boldsymbol{X})$和 $H(\boldsymbol{Y})$分别表示 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 的香农熵.NMI 的取值区间为[0,1],值越高说明聚类质量越好.

对 SCoS 算法和其他并行聚类算法进行 5 趟测试,得到 NMI 结果.测试聚类效果如表 2、表 3 所示.

表 2 聚类算法 NMI 指标对比

算法	MNIST	2-NG	20-NG
SCoS	**0.448(±0.027)**	0.844(±0.011)	**0.411(±0.016)**
ASCoS	0.286(±0.122)	0.812(±0.094)	0.308(±0.125)
Spark k-means	0.403(±0.014)	**0.860(±0.014)**	0.339(±0.026)
Spark PIC	0.255(±0.177)	0.860(±0.100)	0.252(±0.116)

表 3 聚类算法 NMI 指标对比(稀疏化处理,$t=200$)

算法	MNIST	2-NG	20-NG
SCoS	**0.604(±0.021)**	0.841(±0.011)	**0.467(±0.014)**
ASCoS	0.486(±0.116)	0.811(±0.082)	0.377(±0.118)
Spark k-means	0.507(±0.014)	**0.860(±0.014)**	0.395(±0.025)
Spark PIC	0.267(±0.157)	0.861(±0.095)	0.282(±0.109)

其中 SCoS 是采用了 ScaLAPACK 并行化求解精确特征向量的谱聚类算法.ASCoS(Approximate SCoS)是采用了迭代求解近似特征向量的谱聚类算法.Spark k-means 是 Spark Mllib 中的 k-means 算法.Spark PIC 是 Spark Mllib 中的快速迭代聚类算法.

表 2 表示相似度矩阵没有经过稀疏化处理时,4 种算法在 3 个测试数据集上的 NMI 结果.表 3 表示经过 t 近邻稀疏化对称处理后相似度信息上得到的 NMI 结果.从结果中可以看出,SCoS 算法相比于其他聚类算法能够获得更好的聚类效果,而且在不同的数据集上均表现出了良好的聚类性能.ASCoS 由于采用了近似的特征向量,其聚类效果不如 SCoS,但是和 Spark PIC 近似聚类算法相比仍具有一定的优势.另外,相似度矩阵的稀疏化虽然在一定程度上丢失了部分样本之间的相似度信息,但通过表 3 可以看出,经过稀疏化处理后的 NMI 值反而比没有经过稀疏化处理的 NMI 值高,同时,这也说明了通过选取合适的 t 值不仅能够降低大规模相似度矩阵的存储空间开销,而且基本不会影响谱聚类算法的聚类效果.

5.3 算法性能测试

谱聚类算法中,样本之间相似度计算过程和特征向量的求解过程是耗时较长的环节.因此,本文对这两个环节进行了重点测试.

首先,对本文提出的基于多轮迭代的相似度并行计算算法进行测试,并与传统的基于数据集广播的方法进行对比实验.实验数据集选取了 KDD-1、KDD-5、KDD-10、KDD-20、KDD-50、KDD-100.实验结果如图 12 所示.

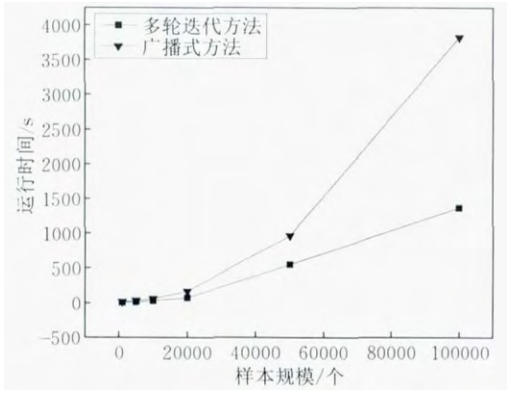

图 12 相似度计算随样本规模增加的运行时间($t=200$)

从图 12 可以看出,基于多轮迭代的方法具有良好的数据可扩展性,性能优于传统的基于数据集广播的方法,而且随着数据规模的增加,基于多轮迭代的相似度计算方法的优势更为明显.多轮迭代的相似度并行计算算法不仅能够保证任意两个样本间的相似度只会计算一次,而且避免了广播式算法由于主节点压力过大容易成为性能瓶颈的问题.

其次,对本文提出的利用 ScaLAPACK 的并行化 MRRR 算法求解正规化 Laplacian 矩阵特征向量进行测试,分析特征向量计算时间随着特征值规模增加的变化情况,并对 SCoS 算法、ASCoS 算法以及 ARPACK 三种求解特征向量的方式进行对比.实验采取了 KDD-20(样本规模 20 000)数据集,特征规模分别设置为 100,200,500,1000,2000.实验结果如图 13 所示.

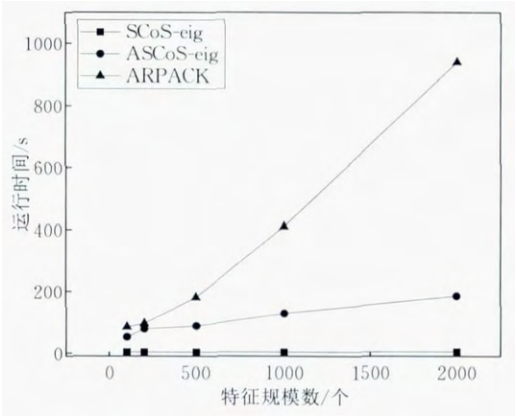

图 13 特征向量计算随特征规模增加的运行时间

从图 13 中可以看出,随着求解特征向量个数的

增加，三种算法的运行时间以不同的速度增长.基于 ScaLAPACK 实现的 SCoS 算法性能最优，其运行时间基本没有明显增长，反映出了通过并行化的 MRRR 算法求解前 k 个特征向量的优势；ASCoS 算法的运行时间呈现出了缓慢的增长，这是因为随着特征规模的增长，ASCoS 算法收敛需要更多的迭代轮数；ARPACK 采用单机求解实现，而且算法复杂度较高[14]，因此随着特征规模增长，ARPACK 运行时间增长较快.

5.4　数据可扩展性测试

为了分析数据规模增长时 SCoS 算法运行时间的变化情况，并测试不同的因素对算法运行效率的影响，本文采用控制变量的方法，分别测试不同样本规模、不同特征/聚类规模情况下算法的运行情况.实验使用 32 个计算核(8 个节点，每个节点 4 个核)，每组数据扩展性实验分别作了 5 趟测试，结果取平均运行时间.

样本规模的数据扩展性实验结果如图 14 所示；不同特征向量/聚类规模的数据扩展性实验结果如图 15 所示.

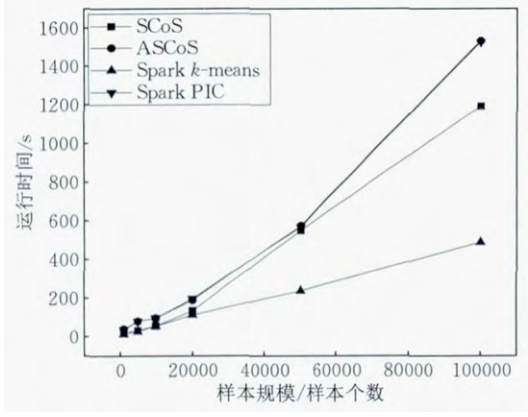

图 14　样本规模增加时算法运行总时间
(特征数：500，聚类数：23，t：200)

从图 14 可以看出，SCoS 算法、ASCoS 算法、Spark k-means 算法以及 Spark PIC 算法均表现出了较好的数据可扩展性.其中，ASCoS 算法和 Spark PIC 算法的运行时间较长，并且运行时间基本相同，这是因为 ASCoS 算法和 Spark PIC 算法都是通过迭代计算实现求解近似特征向量，随着样本规模的增加，ASCoS 算法和 Spark PIC 算法在计算近似特征向量时迭代次数随之增加，算法收敛变慢.Spark k-means 算法在聚类个数固定时，其运算开销主要集中在计算样本与聚类中心的距离.当聚类个数较小时，Spark k-means 算法具有较好的数据可扩展性.SCoS 算法在数据规模增加的情况下，也表现出了较好的数据可扩展性.虽然 SCoS 算法在并行化求解特征向量问题时性能较好，但是随着样本规模的增加，计算任意两个样本之间的相似度的开销也随之增大，所以运行时间要长于 k-means 算法.SCoS 算法、ASCoS 算法、Spark PIC 算法的主要耗时在于相似度计算环节，而 k-means 算法不需要计算两个样本之间的相似度，只需要计算样本与聚类中心的相似度.

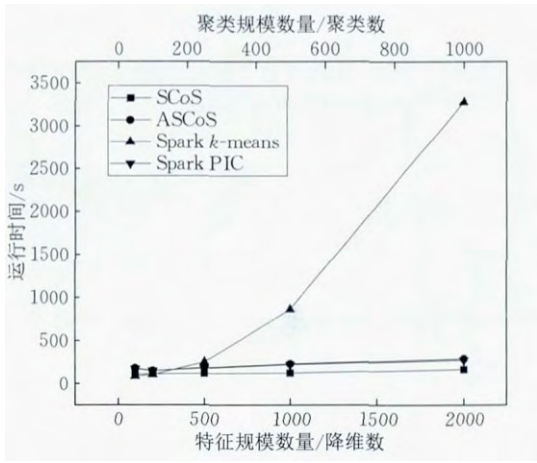

图 15　特征向量/聚类规模增加时算法运行时间
(样本规模：20 000，t：200)

由于本文采取的 KDD Cup99 数据集的样本特征数较少.为了测试 k-means 算法在高维度特征场景下的性能，本文采取了将样本之间的相似度矩阵作为 k-means 算法的输入[39]的方法.从图 15 可以看出，样本规模量一定时，谱聚类算法的优势得到了体现.而 k-means 算法随着输入样本特征数量以及聚类数的增大，计算样本与聚类中心距离的开销以及更新聚类中心点的开销都会增加，导致每一轮迭代的时间变长.SCoS 算法、ASCoS 算法与 Spark PIC 算法由于不受样本输入特征维数的影响，表现出较高的运行效率.整理上来说，随着特征向量数量和聚类数量的增加，SCoS 算法、ASCoS 算法都表现出了较好的数据可扩展性.

5.5　系统可扩展性测试

为了分析 SCoS 算法和 ASCoS 算法随集群规模增大时的性能变化情况，本文又进行了系统可扩展性实验.实验数据集为 KDD-20(样本规模 20 000)，特征向量规模为 500，聚类数为 23.实验中计算节点的规模从 2 个节点增加到 8 个节点(每个节点 4 个核).算法运行时间随节点变化的情况如图 16、图 17、图 18、图 19 所示.

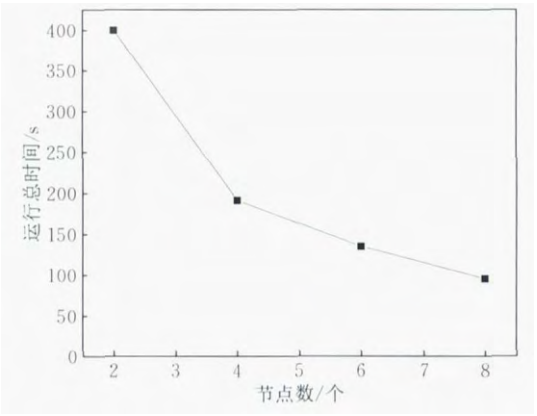

图 16 SCoS 算法随节点数目变化运行时间图（样本规模：20 000，t：200）

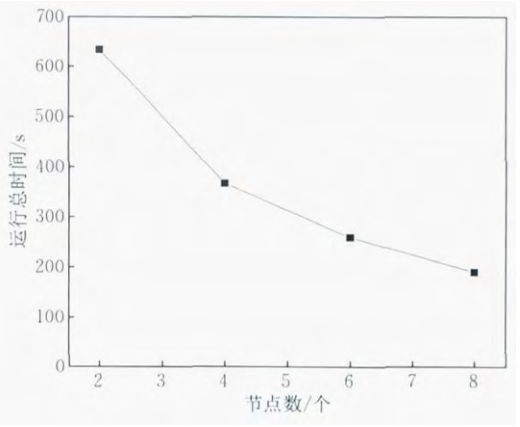

图 17 ASCoS 算法随节点数目变化运行时间图（样本规模：20 000，t：200，聚类数：23）

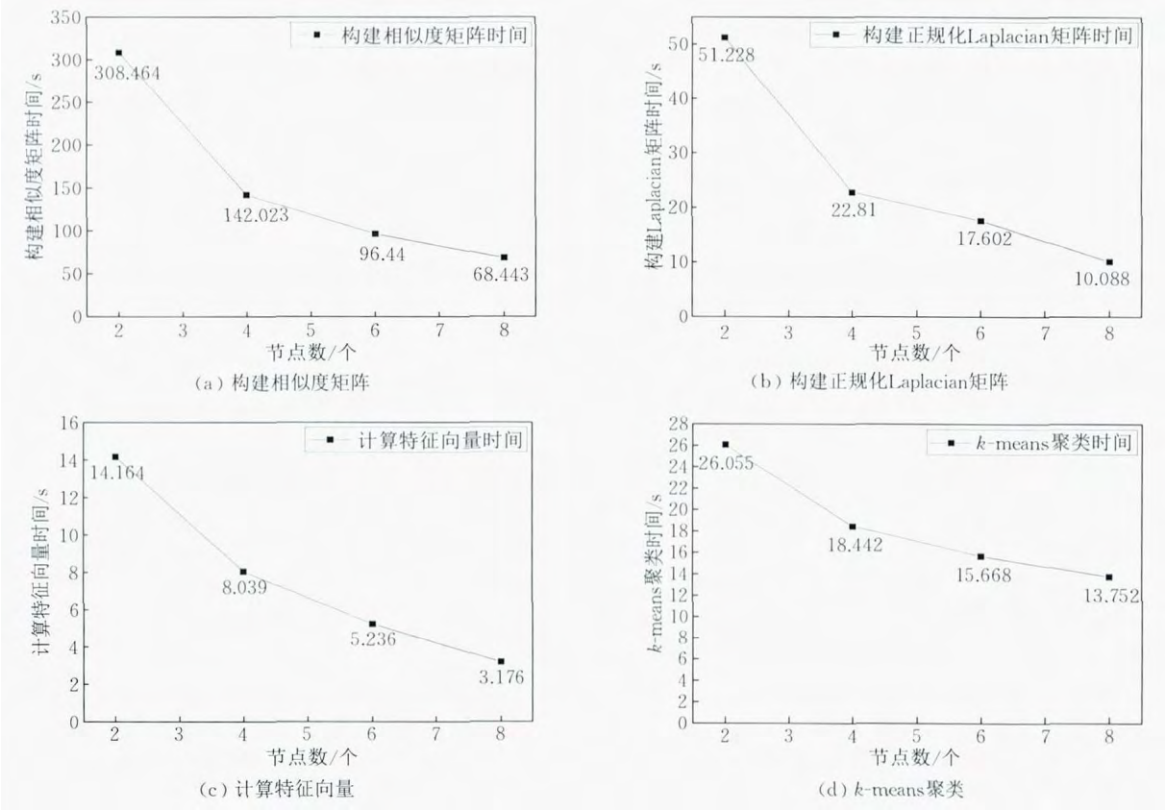

图 18 SCoS 算法各步骤随节点数目变化的运行时间（样本规模：20 000，t：200，聚类数：23）

从图 16 和图 17 可以看出，随着节点个数的增加，SCoS 算法和 ASCoS 算法总体上均呈现较明显的线性下降趋势.

SCoS 算法中各步骤的运行时间随节点数目变化的情况如图 18 所示.可以看出，SCoS 算法中构建相似度矩阵的时间开销占比最大，其随节点变化情况如图 18(a)所示，可以看出，随着节点数目的增加，相似度计算的运行时间呈线性下降的趋势后，逐渐减缓，主要原因是节点间的通信开销逐渐增大.从图 18 可以看出，SCoS 算法中相似度矩阵构建、构建正规化 Laplacian 矩阵构建、特征向量计算以及 k-means 聚类 4 个计算步骤均表现出了较好的系统可扩展性.

ASCoS 算法中的相似度矩阵构建与正规化 Laplacian 矩阵构建步骤和 SCoS 算法一致，但是在求解特征向量环节中，SCoS 采用了 ScaLAPACK 的并行化特征问题求解方法，而 ASCoS 是采用迭代计算的方式实现近似特征向量的求解，迭代计算过程的时间开销如图 19(a)所示，由于迭代收敛过程需要较多的轮数，因此相比于 Spark＋ScaLAPACK

的求解方式，时间开销较大.图 19(b)为 ASCoS 算法在 k-means 聚类过程中的计算开销，由于 ASCoS 是对向量进行 k-means，其时间开销要小于 SCoS 的 k-means 过程.整体来说，SCoS 算法和 ASCoS 算法的运行时间随着节点数目的增加而减少，算法具有很好的系统可扩性.

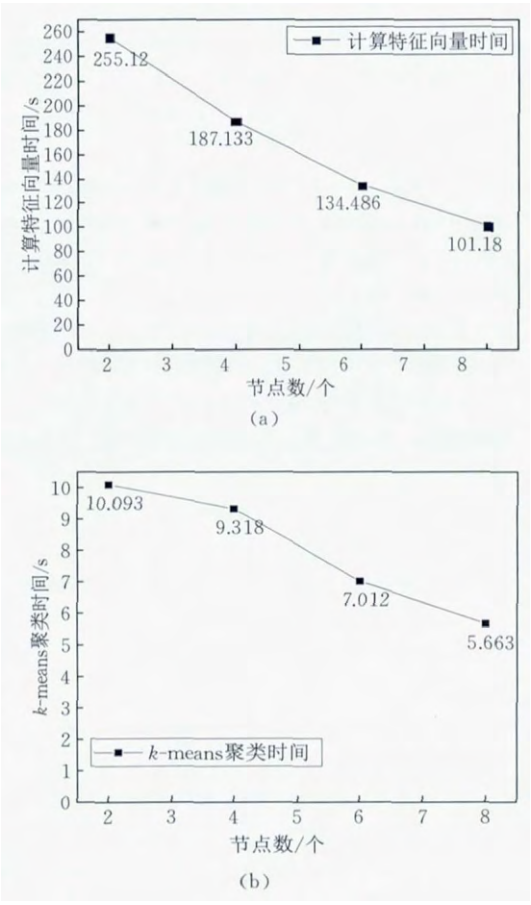

图 19　ASCoS 算法运行时间随节点数目变化（样本规模：20 000，t：200）

6　总　结

针对大规模数据集下谱聚类算法存在的计算开销过大以及计算时间过长等问题，本文基于 Apache Spark 研究并设计了并行化谱聚类算法 SCoS，主要包括相似度矩阵构建与稀疏化过程的并行化、Laplacian 矩阵构建与正规化过程的并行化、正规化 Laplacian 矩阵特征向量计算的并行化以及 k-means 聚类的并行化.在相似度矩阵构建过程中，本文设计并实现了基于多轮迭代的样本间相似度并行计算算法，保证了每两个样本之间相似度只会计算一次，避免了重复计算.另外针对大规模谱聚类算法中耗时较长的 Laplacian 矩阵特征向量求解问题，本文基于 ScaLAPACK 采取了更为高效的精确特征向量并行计算算法 MRRR，同时也实现了近似特征向量计算算法 ASCoS，并且对比分析了精确特征向量计算与近似特征向量计算对于谱聚类算法的性能影响.最后，针对不同的测试数据集，完成了算法的正确性实验以及扩展性实验.实验表明，SCoS 算法不仅具有良好的聚类效果，而且在大规模数据集下表现出了良好的数据和系统可扩展性.

在未来工作中，一方面可以进一步改进 SCoS 算法的性能，使其能够支持 t 近邻数、特征向量数及聚类数的自动调优.另外，尝试将本文实现的 SCoS 算法应用到更多的实际大规模数据场景中，充分发挥谱聚类算法在聚类性能上的优势.

参　考　文　献

[1] Jain A K, Murty M N, Flynn P J. Data clustering: A review. ACM Computing Surveys, 1999, 31(3): 264-323

[2] Fiedler M. Algebraic connectivity of graphs. Czechoslovak Mathematical Journal, 1973, 23(2): 298-305

[3] von Luxburg U. A tutorial on spectral clustering. Statistics and Computing, 2007, 17(4): 395-416

[4] Dhillon I S. Co-clustering documents and words using bipartite spectral graph partitioning//Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining. San Francisco, USA, 2001: 269-274

[5] Shi J, Malik J. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22(8): 888-905

[6] Zelnik-Manor L, Perona P. Self-tuning spectral clustering// Proceedings of the Neural Information Processing Systems. Vancouver, Canada, 2004: 1601-1608

[7] Liu R, Zhang H. Segmentation of 3D meshes through spectral clustering//Proceedings of the Computer Graphics and Applications. Seoul, Korea, 2004: 298-305

[8] White S, Smyth P. A spectral clustering approach to finding communities in graph//Proceedings of the SIAM International Conference on Data Mining. Newport Beach, USA, 2005: 76-84

[9] Dean J, Ghemawat S. Simplified data processing on large clusters//Proceedings of the Operating Systems Design and Implementation. San Francisco, USA, 2004: 107-113

[10] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing//Proceedings of the USENIX Conference on Networked Systems Design and Implementation. San Jose, USA,

2012: 2

[11] Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce//Proceedings of the IEEE International Conference on Cloud Computing. Bangalore, India, 2009: 674-679

[12] Kumari S, Maheshwari A, Goyal P, et al. Parallel framework for efficient k-means clustering//Proceedings of the Annual ACM India Conference. Ghaziabad, India, 2015: 63-71

[13] Chen W Y, Song Y, Bai H, et al. Parallel spectral clustering in distributed systems. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010, 33(3): 568-586

[14] Song Y, Chen W Y, Bai H, et al. Parallel spectral clustering// Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Antwerp, Belgium, 2008: 374-389

[15] Miao G, Song Y, Zhang D, et al. Parallel spectral clustering algorithm for large-scale community data mining//Proceedings of the WWW Workshop on Social Web Search and Mining. Beijing, China, 2008: 1281-1282

[16] Jin R, Kou C, Liu R, et al. Efficient parallel spectral clustering algorithm design for large data sets under cloud computing environment. Journal of Cloud Computing: Advances, Systems and Applications, 2013, 2(1): 18

[17] Chung F R K. Spectral Graph Theory. Rhode Island, USA: American Mathematical Society, 1997

[18] Wu Z, Leahy R. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1993, 15(11): 1101-1113

[19] Hagen L, Kahng A B. New spectral methods for ratio cut partitioning and clustering. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1992, 11(9): 1074-1085

[20] Ding C H Q, He X, Zha H, et al. A min-max cut algorithm for graph partitioning and data clustering//Proceedings of the IEEE International Conference on Data Mining. California, USA, 2001: 107-114

[21] Ng A Y, Jordan M I, Weiss Y. On spectral clustering: Analysis and an algorithm//Proceedings of the Neural Information Processing Systems. Vancouver, British Columbia, Canada, 2001: 849-856

[22] Wu M, Schölkopf B. A local learning approach for clustering// Proceedings of the Neural Information Processing Systems. Vancouver, Canada, 2006: 1529-1536

[23] Bühler T, Hein M. Spectral clustering based on the graph p-Laplacian//Proceedings of the Annual International Conference on Machine Learning. Montreal, Canada, 2009: 81-88

[24] Li Y, Nie F, Huang H, et al. Large-scale multi-view spectral clustering via bipartite graph//Proceedings of the Association for the Advance of Artificial Intelligence. Austin Texas, USA, 2015: 2750-2756

[25] Chang X, Nie F, Ma Z, et al. A convex formulation for spectral shrunk clustering. arXiv preprint arXiv: 1411, 2014: 161-162

[26] Lu C, Yan S, Lin Z. Convex sparse spectral clustering: Single-view to multi-view. IEEE Transactions on Image Processing, 2016, 25(6): 2833-2843

[27] Tian F, Gao B, Cui Q, et al. Learning deep representations for graph clustering//Proceedings of the Association for the Advance of Artificial Intelligence. Quebec City, Canada, 2014: 1293-1299

[28] Cao S, Lu W, Xu Q. Deep neural networks for learning graph representations//Proceedings of the Association for the Advance of Artificial Intelligence. Phoenix, Arizona, USA, 2016: 1145-1152

[29] Yoo S, Huang H, Kasiviswanathan S P. Streaming spectral clustering//Proceedings of the IEEE International Conference on Data Mining. Helsinki, Finland, 2016: 637-648

[30] Belkin M, Niyogi P. Laplacian eigenmaps and spectral techniques for embedding and clustering//Proceedings of the Neural Information Processing Systems. Vancouver, British Columbia, Canada, 2001: 585-591

[31] Meila M, Shi J. A random walks view of spectral segmentation //Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics. Florida, USA, 2001: 92-97

[32] Wang Ling, Bo Lie-Feng, Jiao Li-Cheng. Density-sensitive spectral clustering. Acta Electronica Sinica, 2007, 35(8): 1577-1581(in Chinese)
(**王玲, 薄列峰, 焦李成. 密度敏感的谱聚类. 电子学报**, 2007, 35(8): 1577-1581)

[33] Chang H, Yeung D Y. Robust path-based spectral clustering. Pattern Recognition, 2008, 41(1): 191-203

[34] Fowlkes C, Belongie S, Chung F, et al. Spectral grouping using the Nystrom method. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(2): 214-225

[35] Dhillon I S, Parlett B N, Vömel C. The design and implementation of the MRRR algorithm. ACM Transactions on Mathematical Software, 2006, 32(4): 533-560

[36] Vömel C. ScaLAPACK's MRRR algorithm. ACM Transactions on Mathematical Software. 2010, 37(1): 1-35

[37] Lin F, Cohen W W. Power iteration clustering//Proceedings of the International Conference on Machine Learning. Haifa, Israel, 2010: 655-662

[38] Bahmani B, Moseley B, Vattani A, et al. Scalable k-means++//Proceedings of the International Conference on Very Large Data Bases. Istanbul, Turkey, 2012: 622-633

[39] Karypis G. CLUTO—A clustering toolkit. Department of Computer Science, University of Minnesota, Minneapolis, Minnesota, USA: Technical Report: 02-017, 2002

ZHU Guang-Hui, born in 1987, Ph.D. candidate. His research interests include big data parallel processing and large scale machine learning.

HUANG Sheng-Bin, born in 1993, M.S. His research interests include big data parallel processing and graph computing.

YUAN Chun-Feng, born in 1963, professor. Her research interests include computer architecture, parallel computing, multimedia processing and Web information retrieval and mining.

HUANG Yi-Hua, born in 1962, professor. His research interests include big data parallel processing and cloud computing, computer architecture and parallel computing.

Background

Spectral clustering is an important data analysis technology in data mining and has been widely used in areas such as statistics, image processing, information retrieval, biology, machine learning, etc. Compared with traditional clustering algorithms such as k-means algorithm, spectral clustering algorithm can not only achieve better clustering effect, but also converge to the global optimal solution. However, the large computation overhead is a big problem of spectral clustering algorithm. In some big data scenarios, the computation time is too long to complete clustering.

In order to improve the performance of spectral clustering algorithm, this paper designs and implements a large-scale parallel spectral clustering algorithm with Apache Spark framework, which is called SCoS. Specifically, the main steps of spectral clustering algorithm, including similarity matrix construction and sparsification, Laplacian matrix construction and normalization, eigenvector computation and k-means clustering are all parallelized in SCoS. To further improve the performance of spectral clustering, many optimization methods are implemented in SCoS, such as parallel similarity computation based on multi-round iteration, sparse matrix representation and storage, parallel eigenvector computation optimization, Laplacian matrix multiplication optimization and k-means distance computation optimization.

The experimental results show that SCoS outperforms traditional clustering algorithm in clustering effect. Furthermore, SCoS still has high computation performance in large-scale datasets and achieves better data and system scalability.

This work is support by the Enterprise Cooperative Research on "Large Scale Software Analysis and System Research", the China National Science Foundation Research Grant (61572250), and the Jiangsu Province Science & Technology Research Grant (BE2017155).