一种基于Hadoop的高效 *K* -Medoids并行算法

王永贵,戴 伟,武 超
WANG Yonggui, DAI Wei, WU Chao

辽宁工程技术大学 软件工程学院,辽宁 葫芦岛 125105
College of Software, Liaoning Technical University, Huludao, Liaoning 125105, China

WANG Yonggui, DAI Wei, WU Chao. Highly efficient parallel algorithm of *K* -Medoids based on Hadoop platform. Computer Engineering and Applications, 2015, 51(16):47-54.

Abstract: In view of the traditional *K* -Medoids algorithm is sensitive to the initial clustering center, slow convergence speed, and in large data environment facing the bottleneck problem of memory and CPU processing speed, through improving the initial center options and replacement strategy of using the Hadoop distributed computing platform combined with parallel random sampling strategy based on Top *K* , realizes a highly efficient and stable *K* -Medoids parallel algorithm, and by adjusting the Hadoop platform, realize the further optimization of the algorithm. Experiments show that the improved *K* -Medoids algorithm not only has a good speedup, the convergence and the clustering accuracy are also improved.
Key words: *K* -Medoids; distributed computation; Hadoop; parallel sampling

摘 要: 针对传统 *K* -Medoids算法对初始聚类中心敏感、收敛速度慢,以及在大数据环境下所面临的内存容量和CPU处理速度的瓶颈问题,从改进初始中心选择方案和中心替换策略入手,利用Hadoop分布式计算平台结合基于Top *K* 的并行随机采样策略,实现了一种高效稳定的 *K* -Medoids并行算法,并且通过调整Hadoop平台,实现算法的进一步优化。实验证明,改进的K-Medoids算法不仅有良好的加速比,其收敛性和聚类精度均得到了改善。
关键词: *K* -Medoids; 分布式计算; Hadoop; 并行采样
文献标志码: A **中图分类号:** TP301 doi:10.3778/j.issn.1002-8331.1406-0183

1 引言

在这个大数据的时代,数据量由TB级升至PB级快速地增长,其速度远超摩尔定律的增长速度[1]。面对海量的数据,如何正确处理这些数据,挖掘出有价值的信息,成为当今主要的研究方向之一[2]。利用聚类算法进行信息的自动分类是处理这些数据的重要方法。

K -Medoids算法和 *K* -Means都是基于划分的聚类算法,此类算法简单,并且容易实现,被广泛应用于科学研究的各个领域[3-4]。但是其对初始聚类中心都比较的敏感[5],并且无法很好地处理海量数据的聚类问题。针对 *K* -Medoids算法的特点以及不足,有很多学者提出了各种改进的 *K* -Medoids算法。文献[6-7]实现了基于MapReduce的 *K* -Medoids并行算法,用并行的计算思路解决了处理海量数据的时间问题,但只是针对传统的 *K* -Medoids算法进行并行化处理,没有解决 *K* -Medoids算法对于初始中心敏感的问题;文献[8]根据密度信息产生初始中心,解决了初始中心敏感的问题,但却增加了算法的复杂度。其他的例如利用遗传算法[9],ACO[10],人工蜂群[11]等措施来改进算法,虽然在少量数据环境下能够显著提升 *K* -Medoids算法的性能,但却因为额外的组件增加了算法的复杂性[12],无法适应大数据的现实背景。

因此,针对以上提出的问题,结合当前大数据的环境,本文提出了一种基于Hadoop的高效 *K* -Medoids并行算法,其初始中心的选择使用了基于样本的预处理解决方案,簇心的替换采用了簇内中心调整策略,计算模型选用了MapReduce分布式计算模型,存储系统使用了HDFS(Hadoop Distributed File System)分布式存储系统。实验表明,该算法不仅延续了传统 *K* -Medoids算法的优势,保证了聚类效果的优越性与稳定性,并且在

作者简介: 王永贵(1967—),男,教授,研究领域为云计算、绿色计算、数据挖掘;戴伟(1991—),男,硕士研究生,研究领域为云计算及并行计算技术;武超(1990—),男,硕士研究生,研究领域为云计算及并行计算技术。E-mail:david55555.hi@gmail.com
收稿日期: 2014-06-12 **修回日期:** 2015-01-28 **文章编号:** 1002-8331(2015)16-0047-08
CNKI **网络优先出版:** 2015-01-29, http://www.cnki.net/kcms/detail/11.2127.TP.20150129.1115.009.html

处理海量数据时,算法的执行效率也取得了较大程度的提高。

2 预备知识

2.1 MapReduce编程模型

MapReduce是一种分布式编程模型,其最初是由Google实验室提出的一个分布式并行计算框架,后由Apache组织通过Hadoop分布式计算平台将其开源实现。MapReduce计算模型是将海量的数据进行分割从而进行分布式计算。其对数据的处理抽象成Map和Reduce两个阶段,每个阶段都以键/值(key/value)作为输入和输出,即有[13]:

$$\text{Map}: <\text{key}_1, \text{value}_1> \rightarrow <\vec{\text{key}}_2, \text{value}_2>$$

$$\text{Reduce}: <\text{key}_2, \text{list}(\text{value}_2)> \rightarrow <\vec{\text{key}}_3, \text{value}_3>$$

每一个map以及reduce都会分配到集群中的某一台机器上运行,每一台机器的map或reduce操作都相互独立,这样就实现了map之间的并行,reduce之间的并行,以及map和reduce之间的并行操作过程。

在初始阶段,MapReduce模型通过InputFormat将数据切分成固定大小的片段(split)形成 $<\text{key}_1, \text{value}_1>$ 的数据形式交付给map进行操作。map执行完成以后会根据相同的key合并成 $<\text{key}_2, \text{list}(\text{value}_2)>$ 的形式,然后在Partition中使用Hash算法将相同的key交付给相应的reduce进行处理操作,reduce操作结束后通过OutputFormat形成 $<\text{key}_3, \text{value}_3>$ 的形式存储到文件中。

2.2 Top K 查询算法

Top K 算法的目的就是从海量的数据中选取最大的 K 个元素或记录。其主要的基本思想就是维护一个具有 K 个元素的小顶堆。每当有新的元素加入时,判断其是否大于堆顶元素,若大于则用该元素代替堆顶元素,并重新维护小顶堆,直到处理完所有元素。

3 K-Medoids算法的介绍和分析

3.1 K-Medoids的算法思想

假设有 n 个 h 维的对象点,要将其聚类成 $k(k<n)$ 个簇,则将第 i 个对象点的第 f 维的值定义成 $X_{if}(i=1,2,\cdots,n;f=1,2,\cdots,h)$。这里使用欧式距离来判定对象点之间的相似度,欧式距离越大代表相似度越小。利用欧氏距离,对象 i 与对象 j 之间的距离定义如下:

$$d_{ij}=\sqrt{\sum_{f=1}^{h}(X_{if}-X_{jf})^2}, i=1,2,\cdots,n; j=1,2,\cdots,n \quad (1)$$

传统 K-Medoids算法描述如下:

输入:聚类的个数 k,包含 n 个数据对象的数据集 D。

输出:满足基于各聚类中心对象的距离最小标准的 k 个簇。

步骤1 从数据集 D 中任意挑选 k 个对象作为初始中心点。

步骤2 循环步骤3到步骤5直到每个聚类中心不再变化为止。

步骤3 按照最短距离原则,将对象点分配到离其最近的簇中。

步骤4 全局随机选择一个对象点 O_{random} 作为替代簇心。

步骤5 若替代簇心比原簇心的聚类效果更好,则将替代簇心作为新的聚类中心,否则跳转到步骤2。

3.2 传统 K-Medoids的不足

(1)在进行海量数据的聚类时,由于数据量巨大,无法存储在单台计算机的存储器中,导致 K-Medoids算法无法正常进行聚类操作。

(2)由于 K-Medoids算法自身的特性,其算法的时间复杂度较高,无法适应大数据下的聚类操作。

(3)传统 K-Medoids算法在选择初始聚类中心时采用的是完全随机策略,无法保证聚类结果的准确性。

(4)在聚类中心的替换方法上,传统 K-Medoids采用的是全局顺序替换策略,这种方法虽然保证了聚类的效果,但是却降低了算法的执行效率,增加了算法的执行时间。

4 改进的 K-Medoids算法

4.1 基于Top K 的并行随机采样

面对海量数据,初始中心的选择显得尤其重要,一些改进的算法为了选出较好的初始中心而在聚类操作之前对全局数据进行预处理,但是随着数据量的增大,预处理的代价也逐渐增加,降低了算法整体的运行效率。所以本文采用了先采样,然后在样本内进行数据预处理,计算初始中心。

常用的文本随机采样一般分为两种:(1)遍历采样(2)按字节偏移采样。

第一种遍历采样方法如按行采样,简单并且能够保证原来的数据格式,但由于其遍历过程随着原数据以及采样量的增加,时间复杂度呈线性增长,对系统的消耗将逐渐增大,不适合大规模的采样操作。

第二种按字节偏移的采样方法,能够适应较大规模的数据采样,但是面对海量数据的采样操作,效率也不理想。

针对以上问题,本文提出了一种基于Top K 算法的随机采样过程,并利用Hadoop平台实现了对数据的并行采样,其具体实现过程如下:

输入:随机数范围 H,样本容量 N,Reduce的个数 R_n。

输出: N 条数据样本。

步骤1　在Map阶段给每一个数据随机产生一个0到 H 的整数作为其key值,其数据内容作为value形成<key,value>的键值对输出。

步骤2　将相同key值的数据合并成<key,list<value>>的形式,并根据key值进行内部排序。

步骤3　每个Reduce输出前 N/R_n 条数据。

核心代码如下所示:

(1)Map阶段:

```
Random rd=new Random();
intkey=rd.nextInt(H);
Context.write(new IntWritable(key),value);
```

(2)Reduce阶段:

```
for(Text val:value){
if (i<N/Rn){
  Context.write(null,new Text(val.toString()));
  i++;
  }
}
```

4.2　*K* -Medoids算法的改进方案

(1)基于HDFS的分布式存储策略。因为HDFS是一种分布式存储系统,其将原本需要一整块连续存储空间的数据,拆分成多个小数据块,分别存储在不同的存储节点上。所以在面对海量数据时,也能够对数据进行高效的存取。

(2)针对传统 *K* -Medoids算法的时间复杂度较高,无法进行海量数据的聚类计算问题,本文采用了基于MapReduce的分布式计算模型,将一个大型的数据文件切分成多个小数据模块,分别交付给多台计算机利用MapReduce计算模型进行分布式计算,减少了单台计算机的计算时间,加快了整体的计算速度。

(3)对于初始中心选取的改进一般有两种方案,一种是结合智能算法,第二种则是采用随机策略。由于智能算法在大数据的环境下学习成本较高,而随机选取策略不利于其聚类效果,所以本文结合文献[14-16]提出一种更加适合MapReduce编程模型的样本预处理解决方案,其中采样策略使用了4.1节的并行随机采样,样本容量(N)则根据数据对象个数(M)以及聚类数目(k)进行动态判定。样本容量定义如(2)所示。

$$N=\frac{M}{500}+100\times k \tag{2}$$

数据预处理的计算则使用了(3)中的公式,公式定义如下[9]:

$$V_j=\sum_{i=1}^{n}((\sum_{l=1}^{n}d_{il})-d_{ij}),\ j=1,2,\cdots,n \tag{3}$$

(4)簇内随机替换策略[17-18]。由于顺序替换不能够适应大数据的聚类计算[12],所以采用簇内随机替换策略即在每个簇的内部进行中心点的随机替换过程。实验表明,相比较原来的顺序替换方式,此方法在处理海量数据时有更快的收敛速度。

4.3　新算法描述

改进的 *K* -Medoids算法的具体流程描述如下:

输入:聚类个数 k ,包含 N 个数据对象的数据集,连续次数 C 。

输出:满足基于各聚类中心对象的距离最小标准的 k 个聚类。

处理流程:

步骤1　并行随机采样。

步骤2　样本预处理,计算初始聚类中心,初步聚类:

步骤2-1　利用MapReduce计算模型分布式计算样本内每个节点之间的距离(d_{ij})并保存在相应的文件中,使用的距离测量为公式(1)的欧式距离。

步骤2-2　使用上一步所保存的节点间距离关系,对每个节点 j 利用公式(3)计算 V_j 的值。

步骤2-3　根据 V_j 按照升序进行排序。选取序列中前 k 个值所对应的 k 个点坐标作为初始中心点,写入聚类中心文件,保存。

步骤2-4　按照最短距离原则,将对象点分配到离其最近的聚类中心所在簇中。

步骤2-5　计算各簇内所有节点与其簇心的距离之和(olddistance)。

步骤3　聚类中心替换

各簇内随机选择一个点 O_{random} 替换当前簇心 O_{now} ,计算簇内其他对象点与 O_{random} 距离之和(newdistance),若newdistance较olddistance更小,则将 O_{random} 作为当前簇心 O_{now} ,否则保留原簇心不变。

步骤4　分配节点,判断终止。

步骤4-1　根据当前的聚类中心,将对象点分配到离其最近的聚类中心所在的簇中。

步骤4-2　若 k 个簇心分别连续 C 次不变,则终止算法,否则跳转到步骤3中再次进行。

核心代码如下所示:

(1)KMMapper:map阶段。对象点分配到离其最近的聚类中心所在的簇中

```
setup(){center=簇心文件;}
for (int i=0;i<k;i++){
  for (int j=0;j<point.dimension;j++){
    distance=distance+Math.pow((point(n)j – center[i]),2);
    }
      if (distance<min){
        min=distance;
        p=i;
        }
}
context.write(center[p],new Text(point(n)+":"+min));
```

(2)KMReduce:reduce阶段。选出 O_{random} ,计算

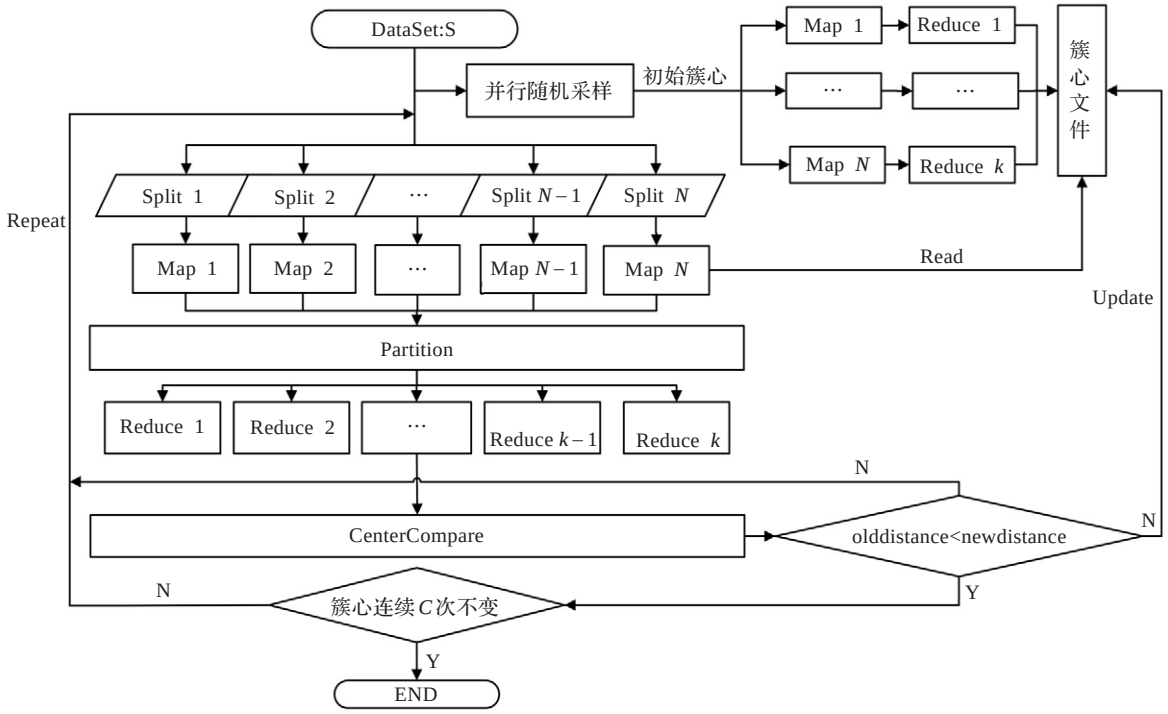

图1　Hadoop 环境下改进的 *K*-Medoids 算法流程图

olddistance 以及 newdistance。

```
while(value.hasNext()){
  olddistance=olddistance+Double.parseDouble(min);
  for (int j=0;j<line.dimension;j++){
    newdistance+=Math.pow((point(n)j − newcenterj),2);
    }
}
context.write(key,new Text(newcenter+""+olddistance+
""+newdistance));
```

(3)CenterCompare:判断终止。比较 olddistance 和 newdistance 的大小,重写簇心文件。

```
for (int i=0;i<k;i++){
  dis=olddistance-newdistance;
  if (dis<=0){
  保留 oldcenter;
  continuationi++;
  }
  else{
  continuationi=0;
  替换 oldcenter,保存 newcenter,重写簇心文件;
  }
}
```

Hadoop 下改进的 K-Medoids 算法流程如图 1 所示。

5　实验与性能分析

5.1　实验环境

在 Linux 环境下搭建 Hadoop 集群,共六台计算机,其中一台作为提供 NameNode,ResourceManager 服务的 master 节点,其他五台作为提供 DataNode,NodeManager 服务的 slave 节点,其中作为 master 节点的配置为:CPU 型号为 Intel core i5-460M;内存为 8 GB;硬盘为 500 GB。其他五台作为 slave 节点的配置为:CPU 型号为 Pentium® Dual-Core E6600;内存为 2 GB;硬盘为 500 GB。六台机器安装的操作系统都为 Ubuntu 12.04,集群上搭建的 Hadoop 版本为 Hadoop-2.2.0。图 2 显示了这六台机器之间的分布关系。

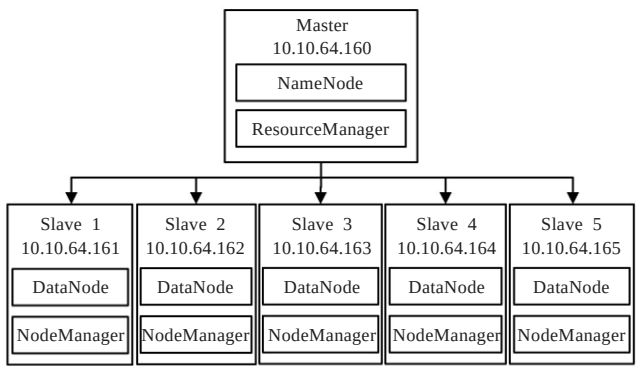

图2　集群结构

5.2　单机处理比较实验

5.2.1　收敛速度比较

实验内容为比较改进的 *K* -Medoids 算法和传统 *K* -Medoids 算法在单机下处理相同数据时,完成聚类所需要的迭代次数及时间。其中改进的 *K* -Medoids 是在 Hadoop 伪分布模式下运行,传统 *K* -Medoids 是在普通串行环境下运行。处理的数据集为 6 类具有 8 206 个对象的三维数据,数据集大小为 0.2 MB,选取的聚类中心 *k* 为 3,具体的收敛时间及迭代次数见表 1。

表1　迭代次数及收敛速度

序号	传统的*K*-Medoids	改进的*K*-Medoids
1	6次/46.089 s	6次/236.945 s
2	9次/72.212 s	6次/253.297 s
3	6次/43.539 s	4次/192.418 s
4	10次/69.735 s	8次/309.424 s
5	7次/51.973 s	7次/313.368 s
6	9次/68.659 s	7次/294.487 s
7	7次/49.296 s	4次/128.298 s

从表1中可以看出,在单机伪分布环境下改进的*K* -Medoids算法平均迭代次数更少,拥有更好的全局收敛性,但是传统*K* -Medoids串行算法所消耗的时间更少。这是因为在处理少量数据时,Hadoop计算平台并不能发挥其性能优势。在MapReduce的计算框架下,不仅要经过提交map和reduce作业的步骤,还要经过将作业分片(split)、中间排序(sort)、合并(merge)、分配(partition)等过程,所以在处理少量数据的情况下,实际的计算时间在总消耗时间中所占比例较小,只有在大量数据的条件下才能更好地发挥出MapReduce计算框架的优势。

5.2.2　单机数据负荷测试

实验内容为在不同数据负荷下,单机伪分布下改进的*K* -Medoids算法和传统串行*K* -Medoids算法执行效率的比较,实验结果如表2,其中*T*1代表传统*K* -Medoids算法所消耗的时间,*T*2代表改进的*K* -Medoids算法所消耗的时间。

从表2可以看出,只有在数据量较小时,串行执行的时间要少于伪分布下的执行时间,并且在数据量达到400 MB以上时,串行算法出现了内存溢出,导致算法无法正常运行,而在Hadoop平台下改进的*K* -Medoids算法却能继续执行。

表2　单机负荷情况

数据集/MB	样本数	*T*1/s	*T*2/s
0.155	15 000	26.581	189.307
1.5	150 000	3 612.398	458.285
93	9 000 000	>7 200	1 207.032
496	47 999 851	Out of Memory	5 328.274

因为传统*K* -Medoids算法聚类时,数据需要读入内存中进行操作,随着数据量的增加,算法的时间复杂度也在急剧增长,所以在面对海量数据的时候,传统的*K* -Medoids算法会导致内存溢出的情况发生。而在MapReduce分布式计算框架下结合HDFS分布式存储系统的支持,将计算及存储过程进行分布式处理,中间数据以<key,value>流的形式传输,降低了单台计算机的负担,所以在Hadoop平台下改进的*K* -Medoids算法也能正常处理海量数据。

5.2.3　Iris数据集对比

本次实验目的是测试在标准数据集下,算法的精确度,所用数据集Iris来自UCI Repository(http://archive.ics.uci.edu/ml/datasets/Iris)。Iris数据集被分成了三组(Setosa,Versicolor,Virginica),每一组包含50个数据对象,每个对象都含有4种属性。分别采用*K* -Means,传统*K* -Medoids以及改进的*K* -Medoids算法来测试各算法聚类的效果。这里由于数据量较小,所以不需要采样,直接对全局数据预处理。

表3,表4,表5分别表示*K* -Means,传统*K* -Medoids以及改进的*K* -Medoids算法对Iris的聚类结果。

表3　*K*-Means聚类结果

	Setosa	Versicolor	Virginica
Setosa	33	17	0
Versicolor	0	13	2
Virginica	17	20	48

表4　传统*K*-Medoids聚类结果

	Setosa	Versicolor	Virginica
Setosa	50	0	0
Versicolor	0	40	10
Virginica	0	10	40

表5　改进*K*-Medoids聚类结果

	Setosa	Versicolor	Virginica
Setosa	50	0	0
Versicolor	0	44	6
Virginica	0	6	44

从表中可以得出,*K* -Means的正确率为62.7%,传统*K* -Medoids的正确率为86.7%,改进的*K* -Medoids正确率为92%,其中*K* -Means的正确率最低,改进的*K* -Medoids正确率最高。因为传统的*K* -Medoids和*K* -Means算法初始中心的选取采用的是随机策略,聚类效果随初始中心的变化而波动,并且*K* -Means算法无法排除脏数据的干扰。而改进的*K* -Medoids算法初始中心的选取采用了数据预处理策略,避免了初始中心的随机性,同时*K* -Medoids算法自身的特性也能很好的避免脏数据的干扰。

5.3　集群测试

5.3.1　初始采样测试

实验内容为比较不同随机采样方式的效率。分别使用逐行遍历采样,字节偏移采样以及基于Top *K* 的并行随机采样进行对比。其中并行采样使用的为六台主机所组成的Hadoop集群,其中一台作为NameNode和ResourceManager,其余五台作为DataNode和NodeManager,选用的为1.2 GB的数据集。实验结果如表6。

从表6可以看出遍历采样的效率最差。抽取的样本较少时,字节偏移采样的效率最高,但是随着采样量的增加,其时间消耗呈线性增长。并行采样所耗费时间趋于平稳,并且在大规模数据采样下,并行采样所耗费的时间最少。所以本文的并行采样方法更适合大数据环境下的采样操作。

表6　不同采样所耗费时间　s

	90	900	900 000	9000 000
逐行遍历	2262.752	>3 600	>3 600	>3 600
字节偏移	1.167	11.062	>3 600	>3 600
并行采样	36.368	36.475	43.912	62.453

5.3.2　人造数据集对比

实验内容为在集群环境下,比较分布式 *K* -Means 算法和改进的 *K* -Medoids 算法的聚类效果。

其中数据集使用的是四组服从二维高斯分布的人造数据集(A:红色实心圆,B:绿色空心圆,C:蓝色正三角,D:蓝绿色五角星),每一组数据包含7 000条二维坐标,其中图3为四组数据所组成原始数据集,图4,图5分别显示的为使用分布式 *K* -Means 算法和改进的 *K* -Medoids 算法的聚类结果。

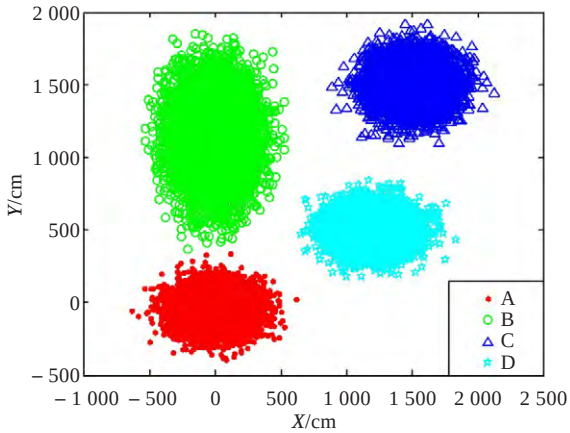

图3　原始数据集

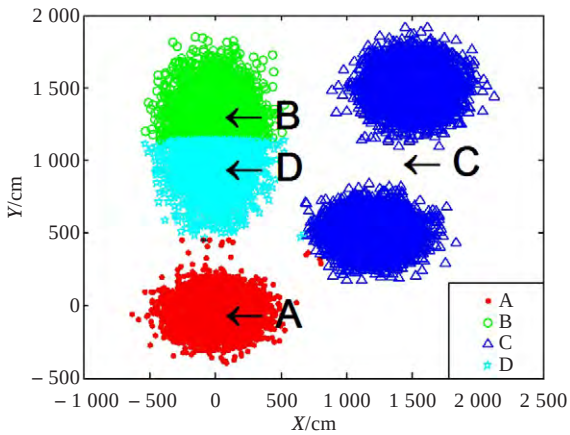

图4　分布式 *K*-Means 聚类效果

图中箭头所指的点即为聚类后的聚类中心点。从图中可以看出,在集群环境下改进的 *K* -Medoids 算法的聚类效果更好,并且其最终的簇心更加接近每一组数据的初始中心。而分布式 *K* -Means 算法的聚类效果却不理想,其A组数据虽然比较接近原始数据,但是B、C、D三组数据的聚类结果都有大幅度的偏差,并且C组数据的簇心指向的是原数据集中不存在的点。这是因为 *K* -Means 是根据均值来计算簇心,无法排除脏数据的干扰,导致其最终簇心可能指向的是原本不存在的点,并且由于其初始中心的随机性,不能保证聚类结果的准确性。所以在集群环境下,改进的 *K* -Medoids 算法的聚类效果更为精确。

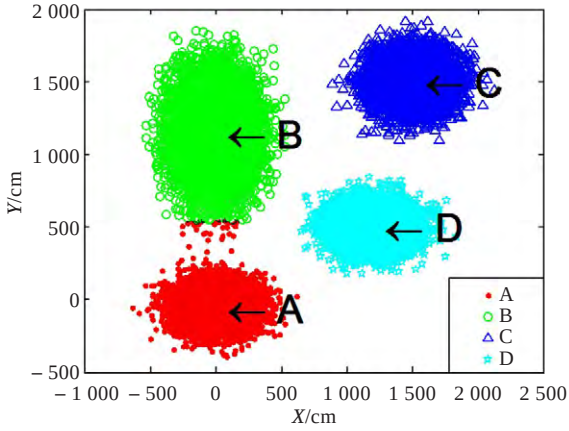

图5　改进的 *K*-Medoids 算法聚类效果

5.3.3　集群负荷测试

实验内容为在Hadoop集群环境下,比较在不同数量的计算节点下,系统处理相同规模数据的效率。表7描述了几组不同的实验数据集。每条记录由2维数值型数据组成,程序指定生成3个中心 ($k=3$),默认的数据分块大小为128 MB。

表7　数据集情况

数据集	数据集大小	记录的数目	块数
A	93 MB	9 000 000	1
B	155 MB	14 999 733	2
C	496 MB	47 999 851	4
D	1.2 GB	119 999 613	10
E	2.8 GB	270 000 987	23

由于实验环境是由5个DataNode所组成的集群,所以分别选择1,2,3,4,5个DataNode节点参与运算,观察在不同数量节点下,系统完成任务的时间,具体的实验结果如图6所示。

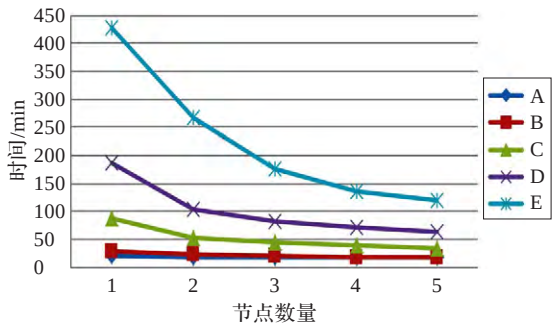

图6　不同节点运行时间图

从图6可以看出,在数据量较小时,不同数量节点下的时间消耗趋于平稳,而当数据量较大时,随着节点的增加,其收敛所消耗的时间明显减少,说明改进的算法更适合应用于大数据下的聚类操作。

5.3.4　集群环境下的加速比

加速比(speedup)，是同一个任务在单处理器系统和并行处理器系统中运行消耗的时间比率，用来衡量并行系统或程序并行化的性能和效果[8]。其计算公式定义为：

$$S_p = T_s / T_p \tag{4}$$

其中 T_p 表示并行算法执行所消耗的时间，T_s 表示串行算法执行所消耗的时间，加速比越大，表示并行算法的执行效率越高。这里实验仍然使用5.3.3节中的数据集，生成4个类别，分别采用1，2，3，4，5个DataNode节点计算其加速比，结果如图7所示。

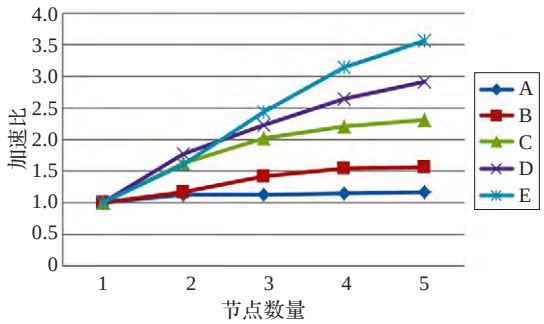

图7　加速比

从图中可以看出每个作业的加速比都随着节点数的增加而增加，尤其当数据量较大时，增加节点可以显著地提高并行执行过程。这说明了通过MapReduce并行计算模型来执行改进的 *K* -Medoids算法可以有效地提高算法的执行效率。

5.3.5　基于Hadoop平台的算法优化

Hadoop中数据是以块(block)为单位进行分布式处理，通常块的个数决定了可以并行的map个数，所以在不同块大小时，算法执行的效率也不相同。在Hadoop中块大小的计算公式定义如下[19]：

$$\max(\text{minimumSize}, \min(\text{maximumSize}, \text{blockSize})) \tag{5}$$

其中minimumSize默认值为1，maxmumSize默认值为Long_MAXVALUE，所以可以通过修改blockSize参数的值来改变块的大小，从而决定map的个数。

这里通过修改blockSize的值，分别将块的大小设置为32 MB，64 MB，128 MB，256 MB来比较改进的 *K* -Medoids算法的执行效率，数据使用5.3.3节中的数据集，其分块情况如表8所示。

表8　不同数据集的分块情况

数据集	数据集大小	32 MB	64 MB	128 MB	256 MB
A	93 MB	3	2	1	1
B	155 MB	5	3	2	1
C	496 MB	16	8	4	2
D	1.2 GB	37	19	10	5
E	2.8 GB	84	42	23	11

不同数据集在不同数据块大小下的运行时间如图8所示，运行环境为6台机器所组成的Hadoop集群，其中5台作为计算节点。

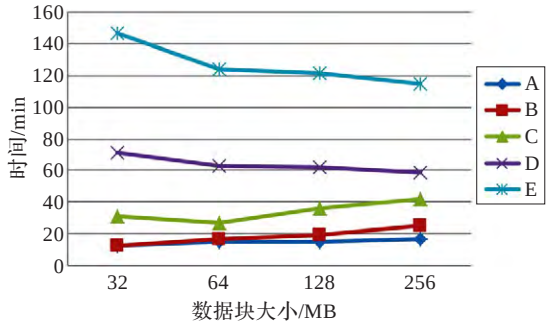

图8　不同块大小的执行效率

可以看出算法执行的效率并不完全和数据块的大小呈正向或反向的线性关系。当数据量较小时，blockSize越大，相应的map个数越少，执行的效率越低；数据量较大时的结果则相反。同时图中显示，数据集 *C* 的时间趋势呈一个倒三角，当blockSize为64 MB时，其执行效率最高。

这是因为当处理的数据量较小时，较大的blockSize会减少可以并行的map个数，无法体现集群并行计算的优势；当处理较大数据时，如果blockSize设置较小，虽然充分利用了集群的并行计算，但却增加了map任务的分配过程以及运行作业所必须的寻址次数，增加了系统的总消耗，尤其在处理海量数据时，这些消耗是相当巨大的。所以在使用Hadoop平台处理数据时，需要根据数据量来合理地进行数据块大小的设置。从以上实验可以得出：在当前集群环境下，数据集和数据块大小设置地合理比例为10∶1。

6　总结

本文在对传统 *K* -Medoids算法研究的基础上，提出了一种并行的高效 *K* -Medoids改进算法，并在Hadoop平台上成功实现。通过标准数据集Iris以及人造数据集的实验结果，可以看出改进的 *K* -Medoids算法比 *K* -Means以及传统 *K* -Medoids算法有着更好的聚类精度。通过在不同大小数据集以及不同数量计算节点下的实验，表明了改进的 *K* -Medoids算法在处理海量数据时较传统的 *K* -Medoids算法有更好的收敛速度，更加适合大数据的聚类计算。最后在Hadoop并行计算平台下，根据其计算机制，通过调整数据块的大小，实现算法的进一步优化。

当然本文算法还有一些需要改进的地方：在初始化中心节点时，通过保存样本点之间的距离关系，使之后的计算能够反复读取距离值，不需要再次计算，减少了算法的计算量。但是由于没有相关数据库的支持，其数据的存储及读取都是基于文件操作，整体效率较低。所以下一步的目标就是引入HBase分布式数据库[20]，从而提高算法的整体性能。

参考文献:

[1] 王珊,王会举,覃雄派,等.架构大数据:挑战、现状与展望[J].计算机学报,2011,34(10):1741-1752.

[2] 覃雄派,王会举,杜小勇,等.大数据分析——RDBMS与MapReduce的竞争与共生[J].软件学报,2012,23(1):32-45.

[3] 孙胜,王元珍.基于核的自适应 *K* -Medoids聚类[J].计算机工程与设计,2009,30(3):674-675.

[4] 孟颖,罗克,刘建华,等.一种基于差分演化的 *K* -medoids聚类算法[J].计算机应用研究,2010,29(5):1651-1653.

[5] Zhang Qiaoping, Couloigner I.A new and efficient *K* -medoid algorithm for spatial[C]//Computational Science and its Applications-ICCSA,2005:181-189.

[6] 张雪萍,龚康莉,赵广才.基于MapReduce的K-Medoids并行算法[J].计算机应用,2013,33(4):1023-1025.

[7] 冀素琴,石洪波.基于MapReduce的*K*_means聚类集成[J].计算机工程,2013,39(9):84-87.

[8] 姚丽娟,罗可,孟颖.一种新的 *k* -medoids聚类算法[J].计算机工程与应用,2013,49(19):153-157.

[9] 郝占刚,王正欧.基于遗传算法和 *k* -medoids算法的聚类新算法[J].现代图书情报技术,2006,136(5):44-46.

[10] 孟颖,罗可,姚丽娟,等.一种基于ACO的*K*-medoids聚类算法[J].计算机工程与应用,2012,48(16):136-139.

[11] 李莲,罗可,周博翔.一种改进人工蜂群的 *K* -medoids聚类算法[J].计算机工程与应用,2013,49(16):146-150.

[12] 夏宁霞,苏一丹,覃希.一种高效的*K*-medoids聚类算法[J].计算机应用研究,2010,27(12):4517-4519.

[13] 虞倩倩,戴月明,李晶晶.基于MapReduce的ACO-*K*-means并行聚类算法[J].计算机工程与应用,2013,49(16):117-120.

[14] Park Hae-Sang, Jun Chi-Hyuck.A simple and fast algorithm for *K* -medoids clustering[J].Expert Systems with Applications,2009,36(2):3336-3341.

[15] Alper Z G.K-harmonic means data clustering with simulated[J].Applied Mathematics and Computation,2007,184:199-209.

[16] Pei Ying, Xu Jungang, Cen Zhiwang, et al.IKMC:An improved K-medoids clustering method for near-duplicated records detection[C]//International Conference on Computational Intelligence and Software Engineering,2009:1-4.

[17] Cardot H, Cénac P, Monnez J M.A fast and recursive algorithm for clustering large datasets with k-medians[J].Computational Statistics and Data Analysis, 2012, 56:1434-1449.

[18] Qiao Shaoyu, Geng Xinyu, Wu Min.An improved method for K_medoids algorithm[C]//International Conference on Business Computing and Global Informatization, 2011:440-444.

[19] Tom White.Hadoop权威指南[M].周敏奇,译.北京:清华大学出版社,2011.

[20] 强彦,卢军佐,刘涛,等.基于HBase的并行BFS方法[J].计算机科学,2013,40(3):228-231.

(上接5页)

(2)引入混沌映射准则,建立三维搜索空间下的改进的果蝇优化算法,并利用该算法训练支持向量机预测模型并进行参数动态寻优,提高了模型的预测精度和效率。

(3)仿真结果表明,基于PCA-MFOA-SVM回采工作面瓦斯涌出量模型与其他模型相比具有较高的预测精度和拟合泛化能力,且综合性能优于其他模型,可运用于瓦斯涌出量动态预测。

参考文献:

[1] 李润求,施式亮,罗文柯.煤矿瓦斯爆炸事故特征与耦合规律研究[J].中国安全科学学报,2010,20(2):69-74.

[2] 姜文忠,霍中刚,秦玉金.矿井瓦斯涌出量预测技术[J].煤炭科学技术,2008,36(6):1-4.

[3] 戴广龙,汪有清,张纯如,等.保护层开采工作面瓦斯涌出量预测[J].煤炭学报,2007,32(4):382-385.

[4] 李胜,宁志勇,朱小强,等.基于灰色理论预测五阳矿未受采动影响煤层瓦斯含量[J].科技导报,2012,30(32):71-74.

[5] 梁晓珍,宋存义,王依.唐山矿瓦斯涌出量动态预测模型[J].北京科技大学学报,2012,34(3):260-263.

[6] 王晓路,刘健,卢建军.基于虚拟状态变量的卡尔曼滤波瓦斯涌出量预测[J].煤炭学报,2011,36(1):80-85.

[7] 温廷新,张波,邵良杉.煤与瓦斯突出预测的随机森林模型[J].计算机工程与应用,2014,50(10):233-237.

[8] 付华,史冬冬.基于IGA-LSSVM的煤矿瓦斯涌出量预测模型研究[J].中国安全科学学报,2013,23(10):51-55.

[9] 何利文,施式亮,宋译,等.回采工作面瓦斯涌出的复杂性及其度量[J].煤炭学报,2008,33(5):547-550.

[10] 戴宏亮.小波支持向量回归在瓦斯涌出量预测中的应用[J].计算机工程与应用,2010,46(7):15-17.

[11] 韩俊英,刘成忠.自适应混沌果蝇优化算法[J].计算机应用,2013,33(5):1313-1316.

[12] 周松林,茆美琴,苏建徽.基于主成分分析与人工神经网络的风电功率预测[J].电网技术,2011,35(9):128-132.

[13] 李胜,韩永亮.基于MFOA-SVM露天矿边坡变形量预测研究[J].中国安全生产科学技术,2015,11(1):11-16.

[14] 杨书佺,舒勤,何川.改进的果蝇算法及其在PPI网络中的应用[J].计算机应用与软件,2014,31(12):291-294.

[15] 张少帅,杨胜强,鹿存荣,等.基于瓦斯涌出量预测的近距离煤层群开采顺序优化选择[J].中国安全生产科学技术,2011,7(9):60-63.

[16] 徐国祥,杨振建.PCA-GA-SVM模型的构建及应用研究[J].数量经济技术经济研究,2011(2):135-147.