基于 Spark 的大规模文本 k-means 并行聚类算法

刘鹏[1),2)], 滕家雨[1),3)],张国鹏[1),2)],胡延军[1),2)],黄宜华[4)*]

[1)](中国矿业大学物联网（感知矿山）研究中心，徐州中国 221008)

[2)](矿山互联网应用技术国家地方联合工程实验室，徐州中国 221008)

[3)](中国矿业大学信息与电气工程学院，徐州中国 221008)

[4)](南京大学计算机软件新技术国家实验室，南京中国 210023)

摘 要 互联网文本数据量的激增使得对其作聚类运算的处理时间显著加长，虽有研究者利用 Hadoop 架构进行了 k-means 并行化研究，但由于很难有效满足 k-means 需要频繁迭代的特点，因此执行效率仍然不能让人满意。本文研究提出了基于新一代并行计算系统 Spark 的 k-means 文本聚类并行化算法，利用 RDD 编程模型充分满足了 k-means 频繁迭代运算的需求。实验结果表明，针对同一聚类文本大数据集和同样的计算环境，基于 Spark 的 k-means 文本聚类并行算法在加速比、扩展性等主要性能指标上明显优于基于 Hadoop 的实现，因此能更好地满足大规模文本数据挖掘算法的需求。

关键词 k-means；并行化；文本聚类；Spark；RDD；Hadoop；MapReduce

中图法分类号 TP311 DOI 号：

Study of parallelized k-means algorithm on massive text based on Spark

LIU Peng[1),2)], TENG Jia-Yu[1),3)],ZHANG Guo-Peng[1),2)],HU Yan-Jun[1),2)],HUANGYi-Hua[4)*]

[1)](Internet of Things Perception Mine Research Centre, China Univ. of Mining and Technology, Xuzhou 221008, China)

[2)](National and Local Joint Engineering Laboratory of Internet Application Technology on Mine, Xuzhou Jiangsu, 221008, China).

[3)](School of Information and Electrical Engineering, China Univ. of Mining and Technology, Xuzhou 221008,China)

[4)](State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210023, China)

Abstract Due to sharp volume increment of internet text, the processing time of k-means on such data is incredibly lengthened. Though some researchers have adopted classic parallel architectures, such as Hadoop, to improve k-means execution efficiency, the results are still unsatisfactory. The major reason is that most mining algorithms, including k-means, need frequent iteration which is hard to be efficiently handled by classic parallel platforms. A parallelization algorithm of k-means, based on Spark, which is the state-of-the-art big data parallel processing architecture, is proposed. The given algorithm makes full use of in-memory-computing RDD model of Spark so as to well meet the frequent iteration requirement of k-means. Experimental results show that k-means executes much more efficiently in Spark than in Hadoop based on the same datasets and the same computing environments, which specifically reflects in the aspects of speedup ratio, the expansion ratio and others. Therefore the conclusion can be drawn that Spark performs sensibly better than Hadoop on processing big data mining algorithms.

Key words k-means；parallelization；text clustering；Spark；RDD；Hadoop；MapReduce

1 引言

随着互联网信息量的迅猛增加，如何对海量网络文本信息进行有效处理及价值挖掘已成为当今中文信息处理的研究热点之一，其中大规模文本聚类便是一个重要的研究领域。由于在互联网信息海洋中，人们

本课题得到国家高技术研究发展计划(863计划)项目(No.2013AA06A411)，国家科技支撑计划项目（2012BAH12B01,2012BAH12B02），中央高校基本科研业务费(No.2011QNB26)资助。第一作者：刘鹏，男，1973年生，博士，副教授，研究领域为大数据并行处理及其在矿山物联网中的应用，E-mail: liupeng@cumt.edu.cn。通信作者：黄宜华，男，1962年生，博士，教授，研究领域为大数据并行处理算法、系统及应用，E-mail: yhuang@nju.edu.cn。

不会事先对数据进行类别标注，也更无法确定数据应归属的类别总数，这样就很难采用基于类别标注的挖掘算法（如支持向量机）对数据进行处理。而聚类技术[1]不需要知道数据的类别标注，同样也不需要任何训练数据便可以直接对目标数据集进行挖掘处理，其应用更加便捷、高效。在互联网大规模信息挖掘处理中，聚类可应用于文本语义分析、文档相似性分析、语料分类分析及主题分析等多个领域[2][3][4]，在很多算法的预处理阶段，例如文本分割以及多文档文摘抽取，聚类技术也可以发挥重要的作用。但是目前可用的聚类算法基本都仅适于处理小规模的数据，而在当今互联网信息爆炸的背景下，网络数据文档数量呈指数性增长，文本特征空间的维度也急剧增大，这都会严重降低聚类算法的类别划分能力，同时也大大延长了算法的运行时间，这显然不适合实际应用，因此如何能够对大规模文本数据进行快速、有效的并行聚类计算将是一个很有价值的研究方向。

作为传统的并行计算方法，上世纪末出现的 MPI(Message Passing Interface)[5]，本世纪初出现的网格计算[6]等存在开发复杂，扩展性不好等问题，已无法满足日益增长的互联网大规模数据处理的要求。面对挑战，“云计算[7]” 应运而生，MapReduce(MR)即是其最受关注的关键技术之一。MapReduce 是由 Google 公司开发的一个用于大规模数据处理的分布式计算模型[8-10]，具有编程简单，易于扩展，容错性好等特点，极大地简化了集群上的海量数据并行处理实现。除 Google 外，MapReduce 已有多种实现版本，其中最经典的是 Hadoop[10]，它是一个由 Java 语言编写的开源分布式计算框架，提供了以 MapReduce 为核心的编程接口和分布式文件系统 HDFS (Hadoop Distributed File System)，能够处理多至百万个节点和 ZB 量级的数据。

但是越来越多的研究已证明[11-12]，Hadoop的MapReduce计算模型比较简单，适合数据量大但核心计算并不复杂的处理作业，而对于较复杂的计算模型，譬如递归、迭代、嵌套调用等，利用MapReduce实现不但编程复杂，而且处理效率也很难让人满意。下面以迭代计算为例说明MapReduce模型的局限性。

目前为止，我们能看到的MapReduce版本实现都没有针对迭代计算进行相应优化，在运行迭代型作业时，存在以下四个主要问题[12]：

（1）尽管每次迭代所进行的操作都一样，但每一次迭代都是作为独立作业（job）重新进行处理，需要重新初始化和读写、传输数据，这会导致大量不必要的系统开销；

（2）虽然大量数据在迭代循环时很可能是不变的，但在每次迭代时仍会被重新载入和处理，这将浪费掉大量的I/O、CPU资源和网络带宽；

（3）每次迭代都需要一个额外的MapReduce Job用来检测迭代终止条件，这又会消耗掉额外的任务调度、磁盘数据读取和网络传输；

（4）必须在前一次迭代全部结束，输出数据全部写入分布式文件系统，并完成迭代终止条件检测之后，下一次迭代才可以开始，而下一次迭代又需先从分布式文件系统读取数据，这使得数据的读写过于集中，而相对应地，空闲时的系统I/O、CPU和网络带宽等资源则被浪费掉了。

因此MapReduce在进行迭代型数据处理时性能比较低下，而迭代计算在数据处理中是一类非常重要的应用模型，尤其在数据挖掘、信息检索、机器学习等领域，大量算法都是运用多次迭代实现的[13]。作为主要创作动机之一，基于内存计算的新一代并行计算框架Spark[14]力求能很好地解决此类频繁迭代问题。本文提出了基于Spark的k-means本文聚类算法并行设计与实现，并在相同计算环境和数据量的情况下，比较分析Hadoop和Spark环境下的并行k-means文本聚类算法执行效率及性能差异，对并行聚类技术在大规模文本分类中的作用做了些有益的探讨。

接下来内容是这样安排的，首先介绍文本聚类整理流程及k-means串行算法，接着分别设计及实现了基于Hadoop和Spark的k-means并行化算法，最后通过实验在加速比、扩展比等主要性能指标上进行比较并得出实验结论。

2 文本聚类整体流程

为了对文本进行聚类分析，必须首先对语料数据进行预处理，包括分词、停用词过滤、特征降维和文本表示模型等步骤，转化为聚类算法能够处理的向量形式，然后才能进行聚类分析，整个文本聚类流程如

图 1 所示，这其中由于聚类分析需要频繁迭代，因此会占据大部分处理时间，所以本文的重点是研究如何通过并行处理提高聚类分析部分的效率。在此之前，先简要介绍下预处理阶段的工作内容（本文文本聚类预处理使用的工具包为中科院计算所研发的 ICTCLAS 分词系统）。

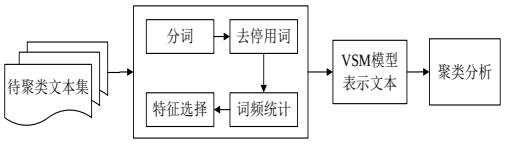

图 1 文本聚类分析整体流程

2.1 文本聚类预处理

文本是非结构化的数据，无法用数据挖掘的方法对文本直接进行处理，需要先对文本进行预处理，使文本最终表示成为一种结构化的计算机能够处理的形式，然后再用聚类算法进行聚类。文本预处理是聚类分析中一个非常重要的环节，预处理的质量直接影响着聚类分析的性能。预处理过程的一般步骤：分词、去停用词、词频统计、特征选择。

预处理的第一个关键步骤就是分词[15]。不同于英文文本，在中文文本中，词与词之间是连续的，没有空格间隔，因此必须对文本进行分词处理。所谓分词，就是将文档按照词的含义进行切分。在文本信息处理的过程中，可以用字、词或者词组作为文本的特征项。但是用字作为特征项会导致特征向量维数庞大，并且字所包含的信息量有限。词组虽然包含信息量多，但是在文本中出现的频率极少，用词组作为特征向量会导致特征向量稀少。因此，选用词作为特征向量即能够包含做够的信息，又能够得到较合适的特征维数。

文本聚类中，从文本得到的单词集还不能作为特征集来表示文本，因为它包含文本集各类文本中普遍出现的通用词和弱词性词，这些停用词几乎出现在任何一个文本中，但是对表达文本内容几乎没有任何贡献，这些词更多的作用是在语法上，被称为停用词。因此，需要建立一个停用表，并按照此表从单词集中过滤所有的停用词，从而降低特征空间维数，减少噪声。

2.2 文本表示

文本是一种无结构的数据，要进行聚类，必须把文本表示成为计算机能够识别和处理的形式。本文采用最常用的向量空间模型（VSM）[16]，向量的每一维由特征项及其权重组成，特征项的权重用 TF-IDF[17][18] 方法来计算：

$$w(t_i,d)=\frac{tf(t_i,d)*\log(\frac{N}{n_i}+0.01)}{\sqrt{\sum_{ti\in d}[tf(ti,d)*\lg(\frac{N}{n_i}+0.01)]^2}} \tag{1}$$

其中：$w(t_i,d)$ 为特征项 t_i 在文档 d 中的权重；$tf(t_i,d)$ 为特征项 t_i 在文档 d 中的词频；N 为训练文本的总数；n_i 为训练文本集中出现特征项 t_i 的文本数，分母为归一化因子，即文档 d_j 的向量化表示为 $d_j=(t_{j1}:w_{j1},t_{j2}:w_{j2},...,t_{ji}:w_{ji},...,t_{jm}:w_{jm})$，即 t_{ji} 表示第 j 个文档的第 i 个特征项，w_{ji} 表示该特征项的权重，m 表示向量中特征项的个数。

3 串行 k-means 文本聚类算法

串行 k-means 聚类算法 [19]基本思想如下：以空间中的 k 个点为聚类中心，对最靠近它们的对象归类，具体计算过程通过迭代的方法，逐次更新各聚类中心的值，直到最后收敛得到最优的聚类结果，其目的就是为了使簇内数据对象之间的相似性尽可能大，而簇间数据对象的相似性尽可能小。在每次迭代后需验证聚类的准则函数是否收敛来确定算法是否应该结束，如果不收敛，就继续对数据对象进行聚类，否则，聚

类完成，算法结束。

运用 k-means 算法进行文本聚类，首先需要对文本建立文本表示模型，向量空间模型是一种常用的文本表示模型。VSM 模型用向量表示文本，文本转换成向量数据，可以利用 k-means 算法实现文本聚类。算法具体描述如下：

输入：文本向量集 $D=\{d_1,d_2,...,d_n\}$，聚类个数 k

输出：k 个聚类

1）从文本向量集 D 中随机选取 k 个向量作为 k 个聚类的初始中心；

2）在第 c 次迭代中，对任意一个向量 d_i，求其到 k 个中心的相似度，将 d_i 归到最相似的类；

3）利用均值方法更新该类的中心值；

4）对所有的 k 个聚类中心，利用上述两步的迭代更新后，求得式 4)的 J 值不再发生明显变化，则判定收敛，迭代结束；或者达到设定的最大迭代次数，迭代也结束。否则转 2）继续迭代。

k-means 算法用来计算文本向量相似度的标准通常是使用欧氏距离，其定义如下：

$$d^2_{(x,y)}=|x_1-y_1|^2+|x_2-y_2|^2+\cdots+|x_p-y_p|^2 \tag{2}$$

其中，$x=(x_1,x_2,\cdots,x_p)$ 和 $y=(y_1,y_2,\cdots,y_p)$ 是数据集中两个 p 维的数据对象。

更新聚类中心的计算方法，定义如下：

$$Z_i=\frac{1}{m}\sum_{x\in C_i}x \tag{3}$$

其中，C_i 表示某个簇，m 表示属于 C_i 簇的数据个数。

评价 k-means 划分聚类效果的聚类准则函数 J 定义为：

$$J=\sum_{i=1}^{k}|\overline{x_i}-o_i|^2 \tag{4}$$

其中：数据划分成的聚类数目为 k，$|\overline{x_i}-o_i|^2$ 为欧氏距离，它表示某个簇类新中心点 $\overline{x_i}$ 到该簇类原中心点 o_i 之间的距离。

4 基于 Hadoop 的 k-means 文本聚类算法并行化

由于 Hadoop 没有针对迭代计算作特殊优化，因此利用 Hadoop 的核心算法 MapReduce 进行 k-means 迭代设计[20]，对原有串行算法中每一次迭代过程都必须对应一次完整的 MapReduce 计算过程，即一个 MapReduce 作业，以完成数据记录到聚类中心的距离计算以及新的聚类中心的计算。我们把每次聚类迭代过程封装成一个 MapReduce 作业 kmeansJob。为了实现聚类过程中的多次迭代计算，需要在聚类主程序中多次创建 kmeansJob。

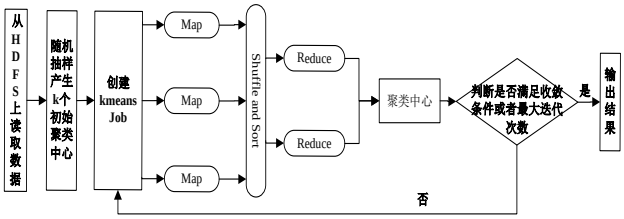

图2 基于Hadoop的k-means并行算法实现总流程图

图 2 描述了聚类主程序首先从 HDFS 上对数据集进行采样并随机生成 k 个中心点，把 k 个中心点作为聚类的初始输入，之后启动相应的 kmeansJob。一次聚类计算包括 Map 和 Reduce 部分，Map 操作执行局部聚类操作，进行数据对象的局部划分聚类， Reduce 操作执行全局的聚类，每个 Reduce 汇总同一个簇的数据点集，计算并更新该聚簇的中心点，具体实现如算法 2-1 所示。当一次聚类计算结束后，根据当前输出结果执行收敛条件判断。如果聚类结果未达到收敛，Hadoop 会再次创建新的 kmeansJob，把上一次 MapReduce 的计算输出作为输入，重复执行聚类任务，直到聚类结果符合收敛条件或者达到最大迭代次数为止。

算法 2-1.kmeansJob 的 MapReduce 化算法

Map 部分

输入：中心点列表，数据对象的偏移量，数据对象

输出：<k1，v1>，k1 为点簇的索引值，v1 为数据点的特征值

FOR 每个文本向量 i=1,…,n do:

FOR 每个聚类中心 j=1,…,k do:

计算每个文本向量与每个聚类中心相似度；

比较上述相似度；

将此向量归纳到相似度高的那个聚类中心所属的类；

将<所属类别，数据对象>写入中间文件；

Reduce 部分

输入：<k2，list[v2]>，k2 为点簇的索引值，list[v2]为数据对象列表

输出：<k3，v3>，k3 为点簇的索引值，v3 为新的中心点

FOR 对于 key 相同的所有文本向量（它们属于同一个簇）：

求所有文本向量的均值，得出新的聚类中心

输出新的聚类中心

5 基于 Spark 的 k-means 文本聚类算法并行化

5.1 Spark 架构和弹性分布式数据集 RDD

Spark 由加州大学伯克利分校 AMPLab 开发，主要目的为用来构建大型的、低延迟的数据分析应用程序，由于引进了弹性分布式数据块 RDD(Resilient Distributed Dataset)[21]的概念，Spark 可在集群计算中将数据集分布式缓存在各节点内存中，省去大量的磁盘 IO 操作，从而大大缩短访问延迟。作为 Spark 架构的核心机制，RDD 是一种基于分布式内存的并行数据结构，它能将用户将数据存储在内存，并控制分区划分以优化数据分布。数据存储在内存中，尤其对于需要多次迭代使用的数据，省去了多次载入到内存和存储到磁盘的过程，大大加快了处理速度。Spark 支持 RDD 的显式缓存(cache)及持久化(persistence)存储，即在第一次使用 RDD 时，开发人员可以显式地调用缓存操作，将所需数据缓存在本地内存，程序结束前，可以调用存储操作将 RDD 持久化到本地文件系统或 HDFS。

Spark 运行架构如图 3 所示，Spark 应用在集群上以独立的执行器(executor)运行在不同节点，在主程序中以 SparkContext 对象来进行总体调度。SparkContext 可以与三类集群资源管理器（Standalone、Mesos 或者 YARN）相连接，本文选择的是 Standalone 模式，集群资源管理器的作用为在不同 Spark 应用间分配资

源。Spark 在执行程序时，需要将应用代码发送给工作节点(worker node)的执行器去执行任务(task)，以尽可能实现数据的本地化计算[22]。

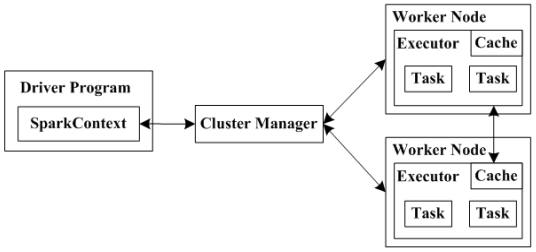

图 3　Spark 运行架构图

与上文所述的 Hadoop 处理迭代运算的方式不同，利用新一代并行计算机架构 Spark 实现 k-means，只需将迭代计算的数据块定义为 RDD，以分区(Partitions)的形式分布存储在不同节点的内存中，再由位于这些节点的 Tasks 针对本地内存 Partitions 重复完成迭代计算即可，中间完全无需和磁盘进行交互，从而大大加快执行时间，这也是以 RDD 内存计算为核心的 Spark 的最大优势。

5.2 基于 Spark 的 k-means 文本聚类并行化设计

利用Spark并行实现k-means，总体上也是采用”map”和”reduce”的思想，即在每次迭代中，先用”map”计算所有样本和中心点距离并归类，再用”reduce”分类求均值算得新的中心点。然而与Hadoop的MapReduce最大的不同是，Spark对所有中心点的所有次迭代运算都是在内存中对RDD计算完成，中间不需与磁盘交互，而Hadoop的这个过程则要与磁盘有（迭代次数*分类数）次的交互。

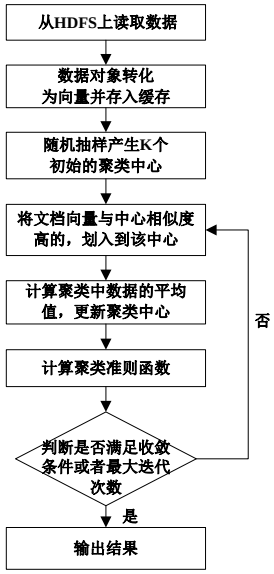

图4基于Spark的K-means算法总体流程图

基于Spark的k-means文本聚类算法在实现时，首先读取HDFS的文件（已经预处理过的文件）并创建新的RDD，并在本地执行Cache操作缓存RDD数据。之后通过Map操作执行局部数据的聚类，Reduce操作执行汇总局部数据的聚类，计算全局的聚簇。聚类算法的并行化执行是由Spark内核调度完成，内核会根据工作节点数目，自动将数据集及执行任务分配到不同节点，继而多个计算节点会并行执行聚类计算。算法总流程如图4所示。

5.3 基于 Spark 的 k-means 文本聚类算法并行化实现

Spark 实现 k-means 并行化算法如算法 3-1 所示，首先执行 Spark 集群环境的初始化；从 HDFS 中载入原始数据，构造 RDD 对象，对 RDD 进行 map 和 cache 操作，生成新的 RDD 对象，将数据转化成特征数据集合，并存于各 worker 节点本地内存；而后在各 worker 节点先执行算法中（1）~（5）完成第一轮迭代，接着重复执行（2）~（5）直到满足收敛条件或者到达最大迭代次数，聚类操作结束。

算法 3-1.基于 Spark 的 k-means 算法

程序输入：Context(arg0)，Path(arg1)，K(arg2)，convergeDist(arg3)，MaxIter(arg4)。其中 Context 为 Spark 环境参数，Path 为数据集输入路径，K 为聚类个数，convergeDist 为聚类准则值 *J*，MaxIter 为最大迭代次数

程序输出：K 个聚类中心。核心步骤如下：

（1）随机抽样产生 *k* 个初始的聚类中心：

$o_1 \sim o_k \in$ 文档向量集D

（2）FOR j=1 to *n* do:

计算所有的 RDD 文本向量与 k 个聚类中心的相似度 $d(x_i, y_j)$，并得出最近的聚类索引号，格式形如（id，（point，1））

（3）FOR *i*=1 to *k* do:

计算各个聚类中所有 RDD 文本向量的平均值，更新聚类中心 $\overline{x_1} \sim \overline{x_k}$

（4）计算聚类准则函数 $J = \sum_{i=1}^{k} | \overline{x_i} - o_i |^2$

（5）until J 不再发生明显变化或者达到最大迭代次数。

Spark 实现 k-means 并行化的代码使用 scala 语言编写，scala 语言表现力强大，代码相比 java、C#等要简洁许多[23]。算法 3-1 所用 scala 主要代码解释如下：1、Spark 集群环境的初始化 2、从 HDFS 读入已处理过的文本向量(3-5)、输入参数赋值给相应变量 6、将每个数据点 RDD 化 7、随机抽样产生 k 个初始聚类中心 (8-9)、聚类准则比较变量和迭代次数中间变量的初始化 10、实现算法 3-1 中（2）~（5）部分，为 Kmeans 核心代码，具体如下：10.0 当聚类准则比较变量大于聚类准则值，并且迭代次数不到最大迭代次数时，执行循环内容｛10.1 求数据的局部聚类，通过 closestPoint 函数求得文本向量 p 与哪个类中心相似度高，标记所属类别，每个点 p 映射成（id，（point，1））。10.2 pointStats 对相同 id 的向量 p 进行归约求和，为（point 特征值求和，point 数量和）。10.3 newPoints 是所求的新聚类中心点，键值对形如（id，point 特征值求和/ point 数量和）。10.4 tempDist 聚类准则变量置零。10.5 新旧中心点的差平方和。10.6 更新聚类中心。10.7 已迭代次数更新。｝

```
1.valsc = new SparkContext(args(0), "SparkKMeans")
2.    val lines = sc.textFile(args(1))
3.    val K = arg(2)
4.    valconvergeDist = arg(3)
5.    valMaxIter = arg(4)
6.val data = lines.map(parseVector _).cache()
7.varkPoints = data.takeSample(false, K, 42).toArray
8.vartempDist = 1.0
9.    vartempIter=0
10.0 while(tempDist>convergeDist&&tempIter<MaxIter)
{
  10.1    var closest = data.map (p => (closestPoint(p, kPoints), (p, 1)))
  10.2 varpointStats = closest.reduceByKey{case ((x1, y1), (x2, y2)) => (x1 + x2, y1 + y2)}
  10.3    varnewPoints = pointStats.map {pair => (pair._1, pair._2._1 / pair._2._2)}.collectAsMap()
  10.4    tempDist = 0.0
  10.5    for (i<- 0 until K){
    10.5.1tempDist += kPoints(i).squaredDist(newPoints(i))}
```

```
  10.6    for (newP<- newPoints) {
     10.6.1kPoints(newP._1) = newP._2}
  10.7tempIter=tempIter+1
}
```

6 实验设计与分析

本实验平台的集群由 1 个控制节点(namenode)和 10 台计算节点(datanode1~10)组成，所有节点配置都相同，有 2 核 CPU（主频：1.86GHz）、4G 内存和 60GB 硬盘，控制节点和计算节点间以千兆以太网互联。Hadoop 的版本为 1.2.1，Spark 的版本为 0.8.1，Hadoop 程序使用 java 编写，java 版本 1.7，Spark 程序使用 scala 编写，版本 2.9.3。本文的语料库采用的是搜狗实验室提供的搜狗语料库，它包括 10 个文本类别集合：汽车，财经，IT，健康，体育，旅游，教育，招聘，文化，军事，数据集的规模如表 1 所示。

表 1 实验使用的数据集

数据集	文件大小
DATA1	188KB
DATA2	18.1MB
DATA3	911MB
DATA4	1.81G

实验分别在 Hadoop 和 Spark 集群平台上进行测试，共进行了 3 类实验：

1） 基于 Spark 平台的并行 k-means 算法加速比测试；

2） 基于 Spark 平台的并行 k-means 算法扩展比测试；

3） 基于 Spark 与 Hadoop 平台，从首次和后续迭代执行时间来比较运行效率。

6.1 k-means 算法的加速比分析

加速比是指通过并行计算使运行时间减少所获得的性能提升，它是验证并行计算性能的一个重要指标，其计算公式为$S_d = T_s / T_d$，其中T_s表示串行算法（即在单节点上）计算所消耗的时间，T_d表示并行算法（即在d个相同节点上）计算所消耗的时间。加速比越大，表明并行计算消耗的相对时间越少，并行效率和性能提升越高。给定数据集 DATA1、DATA2、DATA3、DATA4，指定聚类的个数为 10，测试这些数据集在单机环境下的 k-means 聚类执行的时间及 Spark 环境下的并行聚类算法的执行时间，并行的计算节点为 1 节点、2 节点、4 节点、6 节点、8 节点及 10 节点。图 5 描述的是在 Spark 的环境下的并行算法的加速比，数据集为 DATA1-DATA2 的加速比接近于 1，曲线持平；当数据集为 DATA3-DATA4 时，算法的加速比都大于 1。在相同数量的节点下，数据量很小的情况下，加速比不明显；随着数据量的增长，在相同数据量的情况下加速比曲线随着节点数增加逐渐上升。

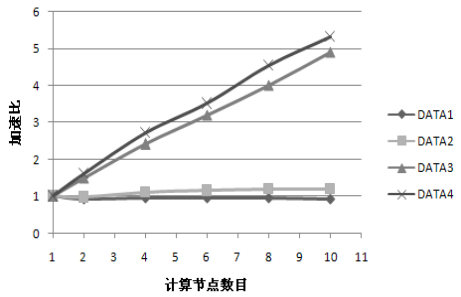

图 5 Spark 环境下的加速比

6.2 k-means 算法的扩展比分析

当集群中的计算节点的数目不断增加时，并行算法的加速比并不能无限地增大，此时仅用“加速比”已

不能反映集群的利用率，因此引入了并行算法效率（扩展比）的概念，即扩展比表示并行算法执行过程中集群的利用率情况，其公式为 $J = S_d / d$，其中 S_d 表示算法的加速比，d 表示计算节点数。图 6 描述了在 Spark 环境下的并行聚类算法执行的扩展比，数据量为 DATA1-DATA2 时扩展比曲线下降很快，而数据集为 DATA3-DATA4 时，并行算法的扩展比曲线下降速率相对趋缓，且随着节点数增加整体趋于平稳。这说明在 Spark 环境下，虽然 k-means 并行算法加速比有明显提高，但随着数据量的增大和节点数量的增多，其扩展比逐渐趋于稳定。

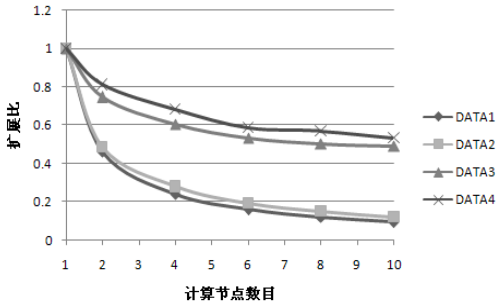

图 6 Spark 环境下的扩展比

6.3 Hadoop 和 Spark 环境下 k-means 并行算法的比较

在本次实验中，每个计算平台的计算节点设置为 10，聚类的数据集合为 DATA4，指定的聚类个数为 10，聚类的阈值 J 为 0.5。我们将分别比较 Hadoop 和 Spark 平台下的算法执行时总的迭代时间及每次迭代的执行时间。实验时随机选择 10 个中心点作为初始中心点。因为 k-means 聚类是典型的迭代算法，我们将第一次迭代（含从磁盘读数据的开销）和除第一次外所有迭代（在 Spark 下，只从内存读数据；Hadoop 下依然每次都要从磁盘读数据）的均值分开描述比较，以更清楚地展现 Spark 基于内存计算的强大优势，如图 7 所示。

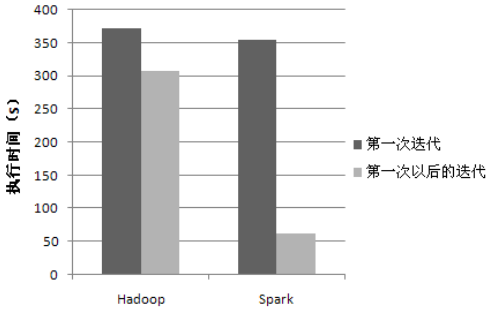

图 7 每次迭代时间比较

在第一次迭代中，两个系统都是从磁盘文件系统 HDFS 中读取数据，如图 6 所示，实验数据显示 Spark 比 Hadoop 稍快，原因是 Hadoop 程序开始和终止的心跳机制开销大。

除第一次迭代外所有迭代运行时间的均值，Spark 比第一次迭代执行时间少了一半，而 Hadoop 与第一次迭代执行时间相差不大。原因是 Hadoop 不支持数据缓存及作业之间的数据共享，而 k-means 的每次迭代都以独立作业的形态存在，因此每次迭代都需要访问 HDFS，将相关的数据集载入到本地内存中；而 Spark 支持数据的缓存，在第一次迭代时会缓存数据到本地内存，在下次迭代计算时执行进程直接从本地内存上读取数据，从而大幅度减少了迭代过程中数据准备的时间开销。因此在后续迭代过程中，Spark 相比 Hadoop 环境下的运行时间得到显著缩减。

图 8 给出了随着计算节点的增加，Hadoop 和 Spark 完成一次迭代的平均执行时间（不含首次迭代）比较，从图中可以看出，相比 Hadoop，Spark 的执行效率得到了大幅度的提升。

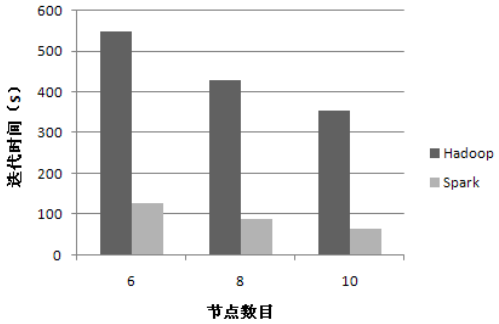

图 8 平均迭代时间比较

7 结语

基于内存计算的新一代并行计算框架 Spark 被人寄予厚望，希望能针对 Hadoop 的弱点，切实提高具有复杂计算模式的大数据文本处理效率。本文以具有频繁迭代计算特点的 k-means 算法并行化为切入点，基于 Spark 平台，利用其以 RDD 为核心的内存计算模型，设计并实现了并行化 k-means 算法。通过实验验证了基于 Spark 的应用在处理需频繁迭代的大规模文本数据计算时，具有优良的加速比和扩展比，与传统大数据平台 Hadoop 相比，处理效率得到显著提升，因此很适合未来大规模数据挖掘及其它需要频繁迭代的数据处理工作。

本文工作受江苏省软件新技术与产业化协同创新中心部分支持。

参考 文 献

[1]何婷婷,戴文华,焦翠珍.基于混合并行遗传算法的文本聚类研究[J].中文信息学报, vol. 21: 55-60, 2007.

[2]陈德华, 韩忠明, 乐嘉锦.基于 XML 文档相似性的构件聚类分析[J].计算机工程与设计, 2009,30(2):507-510.

[3]袁冬.基于海量文本的语义构建方法研究[D].山东：中国海洋大学，2012.

[4]石晶,胡明, 戴国忠.基于小世界模型的中文文本主题分析[J].中文信息学报, vol. 21: 69-74, 2007.

[5] Quinn M J, 奎因, 文光, 等. MPI 与 OpenMP 并行程序设计: C 语言版[M]. 清华大学出版社, 2004.

[6]赵念强, 鞠时光. 网格计算及网格体系结构研究综述[J]. 计算机工程与设计, 2006, 27(5): 728-730.

[7]陈康, 郑纬民. 云计算: 系统实例与研究现状[J]. Journal of Software, 2009, 1(20).

[8] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[9]王凯. MapReduce 集群多用户作业调度方法的研究与实现 [D]. 长沙: 国防科学技术大学, 2010.

[10]刘鹏. 实战 Hadoop: 开启通向云计算的捷径[M]. 电子工业出版社, 2011.

[11]李建江, 崔健, 王聃, 等. MapReduce 并行编程模型研究综述[J]. 电子学报, 2012, 39(11): 2635-2642.

[12]Srirama S N, Batrashev O, Jakovits P, et al. Scalability of parallel scientific applications on the cloud[J]. Scientific Programming, 2011, 19(2): 91-105.

[13]Fox G C. Data intensive applications on clouds[C]. Proceedings of the second international workshop on Data intensive computing in the clouds. ACM, 2011: 1-2.

[14]Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010: 10-10.

[15]黄昌宁, 赵海.中文分词十年回顾[J].中文信息学报, vol. 21: 8-19, 2007.

[16]G. Salton, et al.. Vector-space model for automatic indexing [J]. Communications of the Acm, vol. 18: 613-620, 1975.

[17]徐建民,王金化,马伟瑜.利用本体关联度改进的 TF-IDF 特征词提取方法[J].情报科学，2011,29(2):279-283.

[18]黄承慧,印鉴,侯昉. 一种结合词项语义信息和 TF-IDF 方法的文本相似度量方法[J].计算机学报，2011,34(5):856-864.

[19]韩晓红, 胡彧. K-means 聚类算法的研究[J]. 太原理工大学学报, 2009 (3): 236-239.

[20]江小平, 李成华, 向文, 等. K-means 聚类算法的 MapReduce 并行化实现[J]. 华中科技大学学报: 自然科学版, 2011, 39(1): 120-124.

[21]Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]. Proceedings of the

9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.

[22]Apache Spark. Spark [EB/OL].[2014-02-10]. http://spark.incubator.apache.org.

[23]Scala programming language [EB/OL]. http://www.scala-lang.org, 2014.

Peng Liu(corresponding author: liupeng@cumt.edu.cn) received his Ph.D. from the School of Information and Electrical Engineering at China University of Mining and Technology(CUMT), and his M.Sc. and B.Sc. from the Computer Science Department of Nanjing University, China. He is currently an associate professor in the Internet of Things Research Center, China University of Mining and Technology. Before joining CUMT in 2003, he worked for ZTE Corporation between 1998-2002 as a telecommunication products developer, with the research area of intelligent networks and soft-switch. His current research interests include theory/algorithm/architecture on big-data mining and parallel computing. He has published more than 20 journal and conference papers.