基于 Hadoop 框架的 TF-IDF 算法改进

李 彬
（暨南大学 信息科学技术学院计算机科学系，广东 广州 510632）

摘 要：在分析 Hadoop 框架与 TF-IDF 算法的基础上，给出了 TF-IDF 算法在 Hadoop 分布式框架下的具体实现。实验表明，在处理大数据量时，与传统方法相比，新方法的效率更高。

关键词：Hadoop；TF-IDF；分布式计算
中图分类号：TP311 **文献标识码**：A **文章编号**：1674-7720(2012)07-0014-03

Improve of TF-IDF algorithm based on Hadoop framework

Li Bin
(Department of Computer Science, College of Information Sciencce & Technology, Jinan University, Guangzhou 510632, China)

Abstract: In this paper, carefully analyzed the Hadoop framework and TF-IDF algorithm, give the TF-IDF algorithm based on the Hadoop framework. Experiments show that in the case of massive data computing, the new method applying Hadoop framework is more efficient than the traditional methods.

Key words: Hadoop; TF-IDF; distributed computing

TF-IDF(Term Frequency-Inverse Document Frequency)是一种用于资讯检索与文本挖掘的常用加权技术[1]，用来评估单词对于一个文件集或一个语料库中的其中一份文件的重要程度。单词的重要性随着其在文件中出现的次数成正比增加，但同时会随着其在语料库中出现的频率成反比下降。TF-IDF 算法的各种形式常被搜索引擎、Web 数据挖掘、文本分类及相似度计算等各种应用中，而这些应用往往是以处理海量数据的输入为背景。因此，如何在海量数据中快速有效地计算出 TF-IDF 具有重要意义。

1 TF-IDF 算法原理

在一份给定的文件里，词频 TF(Term Frequency)指的是某一个给定的词语在该文件中出现的次数。对于在某一特定文件里的词语 ti 来说，它的重要性可表示为：

$$tf_{i,j}=\frac{n_{i,j}}{\sum_k n_{k,j}} \quad (1)$$

式中，$n_{i,j}$ 是该词在文件 dj 中的出现次数，而分母则是在文件 dj 中所有字词的出现次数之和。

逆向文件频率 IDF (Inverse Document Frequency)是一个词语普遍重要性的度量。某一特定词语的 IDF，由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到：

$$idf_i=\log\frac{|D|}{|\{j:t_i\in d_j\}|} \quad (2)$$

式中，$|D|$表示语料库中的文件总数，$|\{j:t_i\in d_j\}|$表示包含词语 ti 的文件数目。

在式(1)、式(2)的基础上，可得单词的权重计算公式[2]：

$$tfidf_{i,j}=tf_{i,j}\times idf_i \quad (3)$$

某一特定文件内的高词语频率以及该词语在整个文件集合中的低文件频率，可以产生出高权重的 TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语。

2 Hadoop 简介

Hadoop 是一个开源的可运行于大规模集群上的分布式并行编程框架，它主要由分布式文件系统 HDFS 和 MapReduce 计算模型构成。HDFS 实现了文件的分布式存储，它是 MapReduce 计算的数据载体[3-4]。MapReduce 计算模型的核心是 Map 和 Reduce 两个函数，这两个函数由用户负责实现，功能是按一定的映射规则将输入的<key，value>对转换成另一个或一批<key，value>对输出[4-6]。HDFS 与 MapReduce 的关系如图 1 所示。

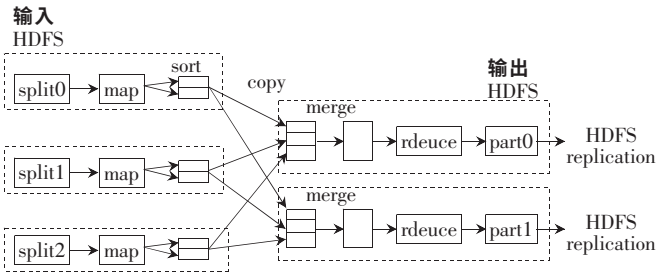

图 1 HDFS 与 MapReduce 的关系

3 Hadoop 框架下的 TF-IDF 算法研究与实现

Hadoop 分布式计算的核心思想是分割任务，并行运行。从 TF-IDF 的计算公式可以看出，它非常适合用分布式计算求解。单词词频 TF 只与它所在文档的单词总数及它在此文档出现的次数有关。因此，可以通过分割数据，并行统计文档中的单词词频 TF，加快计算速度。得到单词词频 TF 后，单词权重 TF-IDF 的计算取决于包含此单词的文档个数（因为文档总数是一个常量）。因此，只要能确定包含此单词的文档个数，即能以并行计算的方式实现 TF-IDF 的求解。本文通过设计 3 次 Map、Reduce 过程实现 TF-IDF 的计算。

3.1 统计每份文档中单词的出现次数

原始数据经过分片后传给 Map 函数。在 Map 中使用正则表达式识别单词，并以键值对<word#documentName，1>的形式写入中间结果，传入 Reduce 函数处理。在 Reduce 中计算单词个数，并将结果输出到临时文件 tempFile1 中以作为下一步 MapReduce 计算的输入。输出结果是以<word#documentName>为键、<n>为值，n 表示单词 word 在文档 documentName 中出现次数。函数设计如下：

Map()：

Input：<documentLineNumer，contents>

Output：<word#documentName，1>

Reduce()：

Input：<word#documentName，1>

Output：<<word#documentName>，n>

此步计算得出单词在文档中的出现次数。

3.2 统计文档单词词频 TF

上一步所得的临时文件 tmpFile1 作为本次 Map 函数的输入。在 Map 函数中，重新组织键值对（以 documentName 为键、<word=n>为值）以便于下一步的 Reduce 计算。Reduce 中，为计算每份文档单词总数，只需累加每份文档的单词数即可。输出结果存入临时文件 tempFile2 中以作为下一步 MapReduce 计算 TF-IDF 的输入。函数设计如下：

Map()：

Input：<<word#documentName>，n>

Output：<documentName，word=n>

Reducer()：

Input：<documentName，word=n>

Output：<<word#documentName>，<n/N>>

此步计算得出每份文档单词词频 TF。

3.3 计算 TF-IDF

以上一步所得的临时文件 tmpFile2 作为 Map 函数的输入。在 Map 函数中，重新组织键值对（以单词 word 为键、<documentName=n/N>为值）以便于计算单词 word 在整个文档集中出现的次数。Reduce 函数中，统计出单词 word 在文档集中出现个数 d、整个文档集个数 D，然后按公式 $TF\text{-}IDF=n/N\times\log(D/d)$ 计算单词的 TF-IDF 值。函数设计如下：

Map()：

Input：<<word#documentName>，n/N>

Output：<word，documentName=n/N>

Reducer()：

Input：<word，documentName=n/N>

Output：<<word#documentName>，TF-IDF>

此步计算得出文档集中每份文档的单词 TF-IDF 值。计算 TF-IDF 的整个处理流程如图 2 所示。

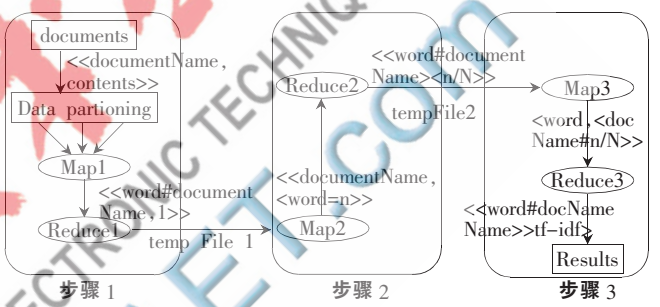

图 2 计算 TF-IDF 的整个处理流程

4 实验与分析

（1）实验数据获取：从（http://www.sogou.com/labs/）提供的文本分类语料库中选取了 20 万篇文档作为实验语料库。

（2）数据预处理：使用开源中文分词工具 IKAnalyzer 对文本进行分词处理，同时去掉停用词。维护过多的小文件会降低 Hadoop 效率，因此需将数据集归档处理。整理后的测试数据如表 1 所示。

表 1 测试数据

数据集	大小/MB	文档个数
S1	40	10 000
S2	80	20 000
S3	160	40 000
S4	320	80 000
S5	520	130 000
S6	780	200 000

（3）Hadoop 群集的搭建：使用了 5 台电脑构建了一个群集，其中一台作为主节点，以负责作业调度及文件空间的管理；其余的作为从节点用作 TF-IDFSS 的计算以及文件的存储。

（4）实验结果：将使用了 Map/Reducer 框架的 TD_IDF 算法与传统的 TF-IDF 计算算法进行对比结果如图 3 所示。传统的 TF-IDF 计算算法只在一台机上运行，且不能以并行和分布式的方式运行。

从图中可以看出，当数据量不大时（<200 MB），传统算法与新算法的差距并不明显。这是因为 Hadoop 本身的维护与网络传输需要一定的开销。随着数据量的增大，传统方法计算 TF-IDF 的时间急剧增长，而应用了 Hadoop 框架的 TF-IDF 计算方法所需时间只是线性增

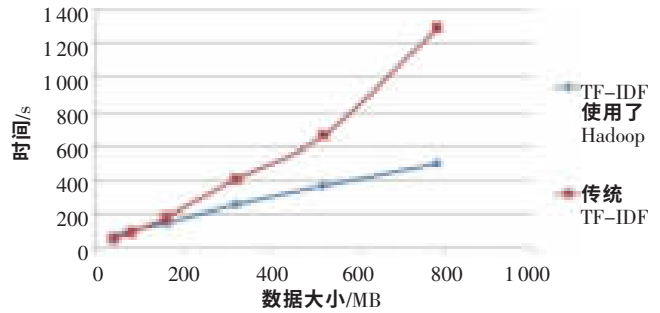

图 3 不同 TF_IDF 计算算法对比图

长，新算法的效率明显高于传统算法。

本文使用了 Hadoop 框架提供的服务改进了计算 TF-IDF 算法的效率。如果仍需要进一步提高算法效率，可以使用序列化文件 SequenceFile 对输入数据做预处理[7]；对 Mapper、Reducer 的输出做压缩；减少中间文件的生成。此外，还可以通过扩展群集规模、改变群集参数等方式来提高效率。

参考文献

[1] SALTON G, BUCKLEY C. Term-weighting approaches in automatic text retrieval [J]. Information Processing and Management, 1988, 24(5):513-523.

[2] SALTON G, CLEMENT T Y. On the construction of effective vocabularies for information retrieval [C]. Proceedings of the 1973 Meeting on Programming Languages and Information Retrieval. New York: ACM, 1973.

[3] The apache softwar foundation HDFS architecture guide[EB/OL]. (2011-11-16) [2011-11-30].Hadoop, http://hadoop.apache.org/hdfs/.do cs/current/index html.

[4] LAMMEL R. Data programmability Team. Google's MapReduce Programming Model-Revisited [R]. Redmond, WA, USA: Microsoft Corp. 2007.

[5] Owen O'Malley. Programming with Hadoop's Map/Reduce [R]. ApacheCon EU, 2008.

[6] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large ousters [C]. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[7] WHITE T. Hadoop: The definitive guide[M]. O'Reilly,2010.

(收稿日期:2011-12-02)

作者简介：

李彬，男，1987 年生，硕士研究生，主要研究方向：计算机应用技术。