云计算期末答辩

李家泽 202214870211 张保营 202214870213 田东旭 202214870129 贾慧爽 202214870206

(华北理工大学，理学院，统计与数据科学系，添砖 Java 组，唐山 063210)

摘　要

本文探讨了基于 Hadoop 分布式计算框架进行大规模文本数据处理的流程。通过搭建 HDFS 和 MapReduce 组件，实现了小说文本数据的分布式存储、分词、词频统计和词云可视化。此外，还利用 LDA 主题模型对小说进行主题分析，提取了潜在的主题特征。研究结果表明，Hadoop 框架能够有效处理大规模文本数据，并结合词云和主题模型，可以更深入地理解文本内容的语义和结构。

关键词：Hadoop、MapReduce、词云图、词频分析、LDA 模型

Abstract

This paper explores the process of processing large-scale text data based on the Hadoop distributed computing framework. By building HDFS and MapReduce components, the distributed storage, segmentation, word frequency statistics, and word cloud visualization of novel text data are realized. In addition, the LDA topic model is used for topic analysis of the novel, and potential topic features are extracted. The research results show that the Hadoop framework can effectively process large-scale text data, and combined with word clouds and topic models, a deeper understanding of the semantics and structure of text content can be achieved.

目录

1 任务描述

本项目的核心任务是基于 Hadoop 分布式计算框架，通过搭建 HDFS、MapReduce 等相关组件，实现一个完整的文本数据处理与分析流程。本项目选择了一部经典小说作为数据源，将其上传至 Hadoop 系统进行分布式存储与处理，最终生成基于小说内容的词云图，以直观展示小说中的高频词汇及其潜在主题特征。

在现代数据处理与分析领域，非结构化数据（如文本数据）已成为极具价值的资源。然而，大规模的文本数据处理通常面临存储和计算的双重挑战。Hadoop 作为一个成熟的分布式计算框架，提供了高效处理大数据的能力，其中 HDFS（Hadoop Distributed File System）是 Hadoop 的分布式存储组件，提供了可靠的数据存储机制，支持大文件的分块存储和容错；而 MapReduce 作为 Hadoop 的核心计算框架，允许将大规模数据处理任务并行化分配到集群中的多个节点执行，并且具备节点故障自动恢复的能力。通过将文本数据存储于 HDFS，并利用 MapReduce 框架实现分布式处理，本项目能够有效地对小说文本数据进行高效分词和词频统计，为后续的词云生成和主题分析提供数据支持。

本项目的实现流程包括以下几个关键步骤。

1. 利用 Docker 搭建 Hadoop 环境，完成 HDFS 和 MapReduce 组件的配置与测试，以确保系统能够支持分布式数据存储和并行化处理任务。

2. 将小说文本数据上传至 HDFS，并利用分词工具对文本进行预处理，包括切分单词、去除停用词和无关词汇，确保词频统计的结果更符合实际需求。

3. 基于 MapReduce 计算框架完成词频统计，并使用统计结果生成词云图。词云图是一种直观的文本可视化方式，词汇的显示大小和颜色与其出现频率成正比，从而突出显示小说中的核心词汇。

4. 基于预处理后的文本进行主题分析，提取隐去小说的作者和标题等敏感信息的核心主题。

本项目的开展有助于加深对 Hadoop 分布式架构的理解，掌握 HDFS 数据存储原理与 MapReduce 计算模型的实际应用，探索在大数据环境下进行文本分析和自然语言处理的技术方法。通过本次实验，我们将深入了解如何在分布式环境中进行数据预处理、词频统计和可视化，结合数据分析结果生成词云图，进一步提升数据分析的展示效果。最终，本项目的成果将以论文形式呈现，内容涵盖系统部署、设计与实现、实验结果及性能分析，以全面展示 Hadoop 在大规模文本数据处理中的应用潜力和优势。

2 Hadoop 部署

2.1 Hadoop 架构

Hadoop 由两个主要组件组成：HDFS 和 YARN，它们实现了上一节讨论的分布式存储和计算的基本概念。HDFS（有时缩写为 DFS）是 Hadoop 的分布式文件系统，负责管理存储在集群中磁盘上的数据；YARN 则是集群资源管理器，将计算资源（worker 节点上的处理能力和内存）分配给希望执行分布式计算的应用程序。

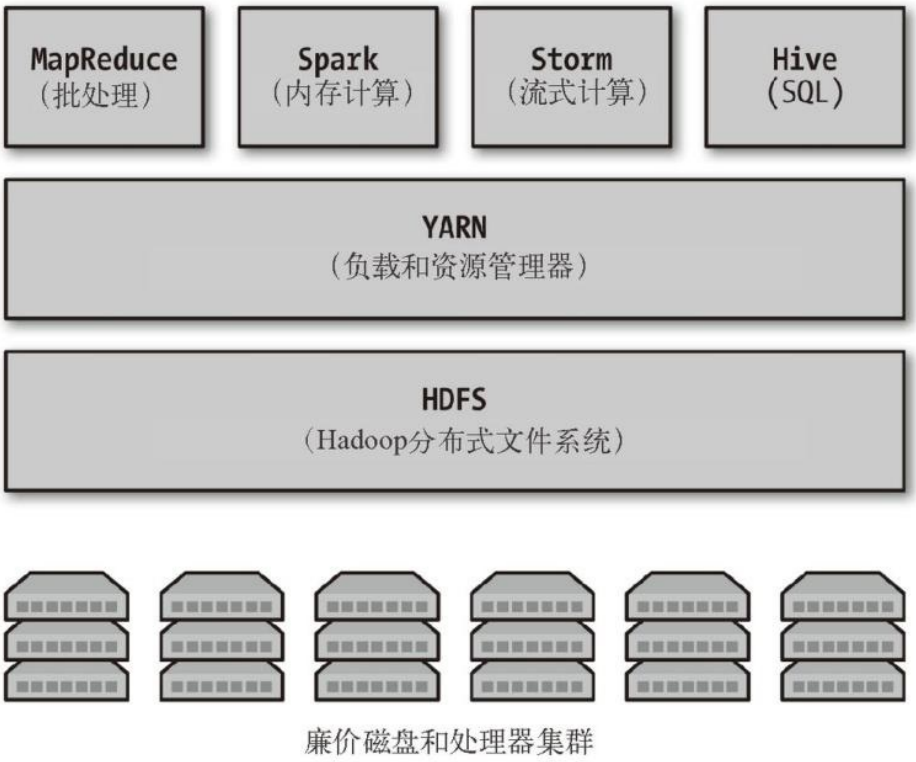

图 1: Hadoop 由 HDFS 和 YARN 构成

HDFS 和 YARN 协同工作，主要通过确保数据对于所需的计算是本地的，最大限度地减少集群中的网络流量。数据和任务的重复确保了容错性、可恢复性和一致性。此外，集群被集中管理，提供了可扩展性，还可将底层的集群编程细节抽象化。HDFS 和 YARN 共同构建了大数据应用程序的平台——也许不仅仅是一个平台，它们为大数据提供了一个操作系统。

HDFS 和 YARN 都有多个 master 服务，负责协调运行在各个 worker 节点上的 worker 服务。worker 节点实现 HDFS 和 YARN 的 worker 服务。HDFS 和 YARN 的 master 服务和 worker 服务，如下所示。

2.1.1 master 节点

这些节点为 Hadoop 的 worker 节点提供协调服务，通常是用户访问集群的入口点。没有 master 节点，协调就不复存在，也就不可能进行分布式存储或计算。

- **NameNode（HDFS）**

  用于存储文件系统的目录树、文件元数据和集群中每个文件的位置。如果客户端想访问 HDFS，必须先通过从 NameNode 请求信息来查找相应的存储节点。

- **SecondaryNameNode（HDFS）**

  代表 NameNode 执行内务任务并记录检查点。虽然它叫这个名字，但它并不是 NameNode 的备份。

- **ResourceManager（YARN）**

  为应用程序分配和监视可用的集群资源（如内存和处理器核心这样的物理资源），处理集群上作业的调度。

- **ApplicationMaster（YARN）**

  根据 ResourceManager 的调度，协调在集群上运行的特定应用程序。

2.1.2 worker 节点

集群中的大多数计算机都属于这类节点。worker 节点运行的服务从 master 节点接受任务——存储或检索数据、运行特定应用程序。worker 节点通过并行分析运行分布式计算。

- **DataNode（HDFS）**

  用于存储和管理本地磁盘上的 HDFS 块，将各个数据存储的健康状况和状态报告给 NameNode。

  从宏观上看，当从 HDFS 访问数据时，客户端应用程序必须先向 NameNode 发出请求，以在磁盘上定位数据。NameNode 将回复一个存储数据的 DataNode 列表，客户端必须直接从 DataNode 请求每个数据块。注意，NameNode 不存储数据，也不将数据从 DataNode 传递到客户端，而是像交警指挥交通一般，将客户端指向正确的 DataNode。

- **NodeManager（YARN）**

  在单个节点上运行和管理处理任务，报告任务运行时的健康状况和状态。

  与 HDFS 的工作方式类似，如果客户端希望执行作业，就必须先向 ResourceManager 请求资源，ResourceManager 会分配一个应用程序专用的 ApplicationMaster，它在作业的执行过程中会一直存在。ApplicationMaster 跟踪作业的执行，ResourceManager 则跟踪节点的状态，每个 NodeManager 创建容器并在其中执行任务。请注意，Hadoop 集群上也可能运行着其他进程（例如 JobHistory 服务器或 ZooKeeper 协调器），但这些服务是 Hadoop 集群中运行的主要软件。

2.2 Docker 简介

Docker 是一种开源的容器化平台，旨在简化应用程序的开发、部署和运行过程。它提供了一种轻量级、可移植和自包含的容器化环境，使开发人员能够在不同的计算机上以一致的方式构建、打包和分发应用程序。

2.2.1 Docker 架构

Docker 包括三个基本概念:

- **镜像（Image）**

  Docker 镜像（Image），就相当于是一个 root 文件系统。比如官方镜像 ubuntu:16.04 就包含了完整的一套 Ubuntu16.04 最小系统的 root 文件系统。

- **容器（Container）**

  镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

- **仓库（Repository）**

  仓库可看成一个代码控制中心，用来保存镜像。

2.2.2 Docker 优点

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势:

- **更高效的利用系统资源**

  由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。

- **更轻松的维护和扩展**

  Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。

- **一致的运行环境**

  开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致，导致有些 bug 并未在开发过程中被发现。而 Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现「这段代码在我机器上没问题啊」这类问题。

表 1: Docker 对比传统虚拟机总结

	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

2.3 Hadoop 集群

本文中所使用的 Hadoop 集群由三个 Docker 容器组成，各 Hadoop 组件运行情况如下表所示。

表 2: Hadoop 集群架构

主机	NameNode	SecondaryNameNode	DataNode	ResourceManager	NodeManager
master	是			是	
worker1		是	是		是
worker2			是		是

2.3.1 关键参数说明

- **hadoop-env.sh**

  Hadoop 运行的环境变量，用于指定 jdk 路径、指定各进程（NameNode 等）的用户名。使用 root 账户运行 Hadoop 时务必指定用户名为 root，否则会报错。

- **core-site.xml**

  Hadoop 核心全局配置，用于指定默认的文件系统及端口、是否开启用户认证等。

- **hdfs-site.xml**

  HDFS 配置，继承 core-site.xml，用于指定 SecondaryNameNode 节点地址、每个 block 的备份数量、数据存储路径等。

- **mapred-site.xml**

  MapReduce 配置，继承 core-site.xml，用于指定 MapReduce 运行所需的环境变量。

- **yarn-site.xml**

  YARN 配置，继承 core-site.xml，用于指定 resourcemanager 节点地址。

- **workers**

  从节点配置，用于指定工作节点，本文中的工作节点为 worker1 和 worker2。

3 小说词频分析

在本研究中，利用 Hadoop 分布式计算框架对小说文本数据进行存储、处理和分析，通过中文文本预处理、词频分析和词云可视化三大部分，主要依托 HDFS 的分布式存储与 MapReduce 的并行计算能力，实现大规模文本数据的高效处理。

3.1 中文文本预处理

3.1.1 分词工具介绍

Jieba 分词器是一款基于 Python 的开源中文分词工具，它采用基于词频统计的隐马尔可夫模型（HMM）实现。Jieba 分词支持三种分词模式：精确模式、全模式和搜索引擎模式，以满足不同的分词需求。Jieba 分词的特点是速度快、准确度高，非常适合用于中文文本的分析。

3.1.2 分词模式选择

在本次研究中，我们选择了 Jieba 分词的精确模式，因为这种模式能够提供更为准确的分词结果，适合进行细致的文本分析。以下是 Jieba 分词的三种模式介绍：

- **全模式**

  将句子中的所有可能的词语都扫描提取出来，速度快但冗余多，不能解决歧义问题。全模式适合搜索引擎索引场景，不适合分析和处理冗余信息的场景。

- **精确模式**

  根据语料库和概率模型找到句子的最优切分方式，尽量将句子分成最精确的词，分词结果准确度高，适用于信息抽取、数据分析等场景。

- **搜索引擎模式**

  在精确模式的基础上，对长词再次进行切分，以提高召回率。搜索引擎模式的分词结果覆盖度更高，适合搜索引擎索引优化和精准匹配场景。

3.1.3 分词的实现及分析

使用 jieba 分词器的命令行分词模式，将连续的文本字符串切分成有意义的词语单元，得到以空格分隔的小说文本，这种分词结果为我们后续的文本分析提供了良好的基础。

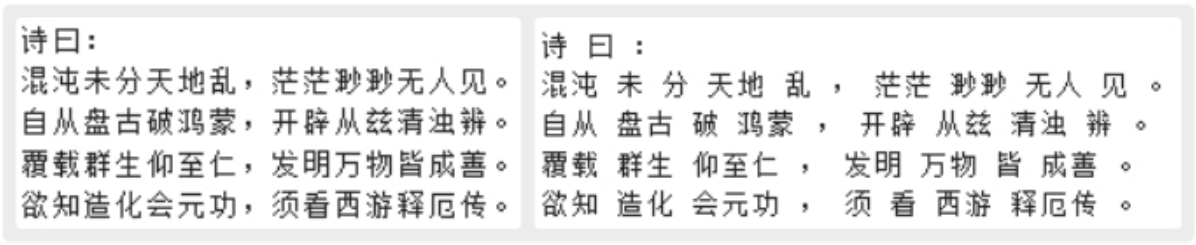

图 2: 分词前后文本对比

3.2 基于 MapReduce 的词频分析

3.2.1 原理

词频分析旨在统计文本数据中每个词语出现的次数。在文本分析中，频繁出现的词语往往与文本的主题相关联。MapReduce 词频分析的核心在于将文本数据分割成多个片段，通过 Map 和 Reduce 两个阶段来并行处理这些数据。

- **Map 阶段**

  在 Map 阶段，将输入的文本数据转换为一系列的键值对。每个键代表一个分词后的词语，而值则是一个计数单位，用于表示该词语的出现次数。

- **Reduce 阶段**

  在 Reduce 阶段，接收来自 Map 阶段的中间结果，并将具有相同键的值进行累加。这一步骤的目标是计算出每个词语在整个数据集中的总出现频率。

3.2.2 实现细节

- **WordCountMapper 类**

  在此类的 setup() 方法中，使用 BufferedReader 读取指定路径的文件，加载停用词并存储在一个 HashSet 中。

  在此类的 map() 方法中，读入一行文本，将其按空格分割成词语，并循环处理每个词语，过滤掉长度小于等于 1 的词语、不符合正则表达式的词语和停用词，留下有效词及其出现次数（1）作为 map 的输出。

- **WordCountReducer 类**

  在此类的 reduce() 方法中，接受来自 map 阶段的输出，遍历所有相同的单词对应的值（出现次数），累加这些值，最终输出词语及其总出现次数。

- **WordCountDriver 类**

  此类用于配置并提交一个 Hadoop MapReduce 任务。

3.2.3 运行结果及分析

运行完成后得到一个文本文件，其每一行是以制表符分隔的键值对，每个键表示一个词语，每个值表示词频，如下图所示。

```
1    一一对    1
2    一七维    1
3    一万 7
4    一万三千五百斤 4
5    一万个    2
```

图 3: 词频分析结果

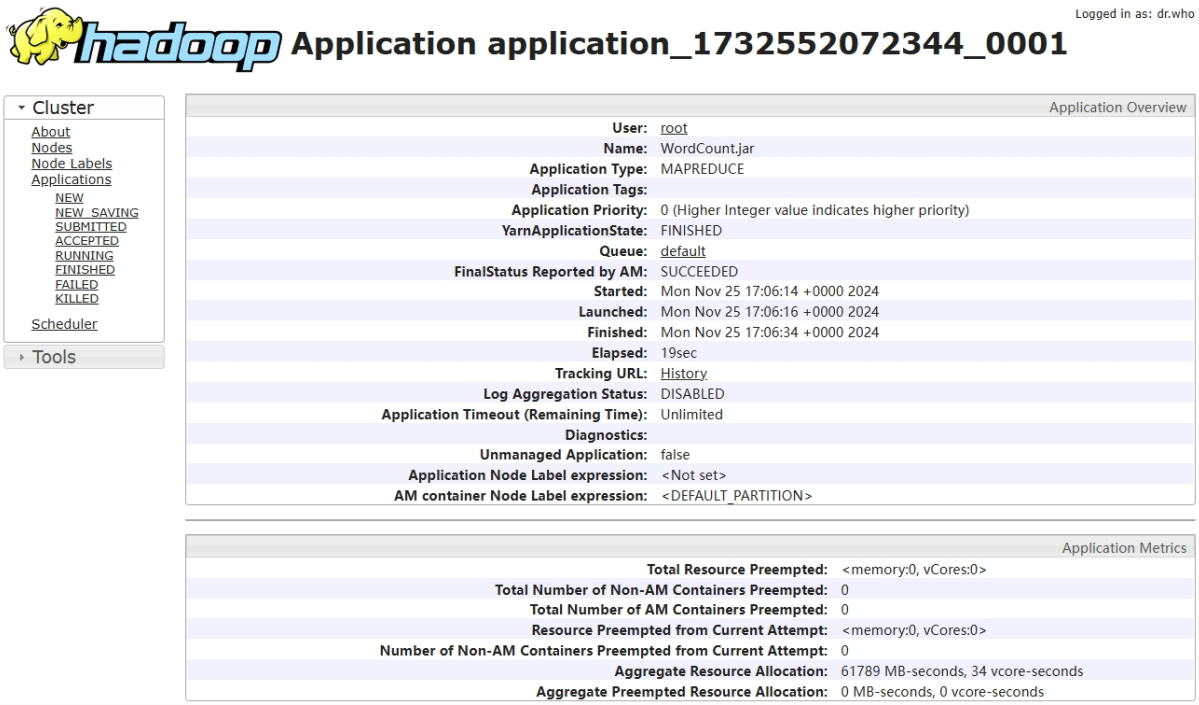

图 4: YARN 运行记录

上图为 WordCount 应用的运行记录，应用在 19 秒内成功完成，表明该任务相对较小或集群资源充足，能够快速处理。

3.3 词云可视化

3.3.1 词云可视化工具介绍

wordcloud2 是一款通过调用一个 JS 的库实现词云图的 R 包，它提供了一个简单易用的接口来生成词云。该 R 包具有以下特点：

- **交互性**：用户可以点击词云中的单词来获取更多信息。
- **灵活性**：用户可以自定义词云的形状、颜色、字体大小等。
- **易于使用**：该包的 API 设计简洁，易于上手。
- **支持多种数据源**：可以接受文本数据、数据框（data frame）等作为输入。

3.3.2 运行结果及分析

使用 R 语言读取上一阶段词频分析的结果，并按词频降序排列数据，最终绘制词云图如下

图 5: 小说词云图

通过词云图我们可以了解到，该小说中出现最多的词云是师父、菩萨、和尚、长老、妖精等词，显示字体较大，说明其在文本中的重要性。

4 小说主题分析

在本研究中，为了进一步深入分析文本数据，我们引入了 LDA 主题模型。LDA 是一种概率生成模型，用于文本数据的聚类分析，可以发现文档集中的潜在主题结构。给出根据主题进行主题聚类或文本分类。

4.1 LDA 模型概述

LDA 主题模型是一种无监督学习算法，主要用于推测文档的主题分布，它假设文档是由一系列的主题组成，而每个主题又由一系列的词汇构成。通过模型训练，可以发现文档集中的潜在主题，并识别出每个主题的主要词汇。

在 LDA 模型中一篇文档生成的方式如下:

- 从狄利克雷分布 α 中取样生成文档 i 的主题分布 θ_i
- 从主题的多项式分布 θ_i 中取样生成文档 i 第 j 个词的主题 $z_{i,j}$
- 从狄利克雷分布 β 中取样生成主题 $z_{i,j}$ 对应的词语分布 $\phi_{z_{i,j}}$

- 从词语的多项式分布 $\phi_{z_{i,j}}$ 中采样最终生成词语 $w_{i,j}$

其中，类似 Beta 分布是二项式分布的共轭先验概率分布，而狄利克雷分布（Dirichlet 分布）是多项式分布的共轭先验概率分布。

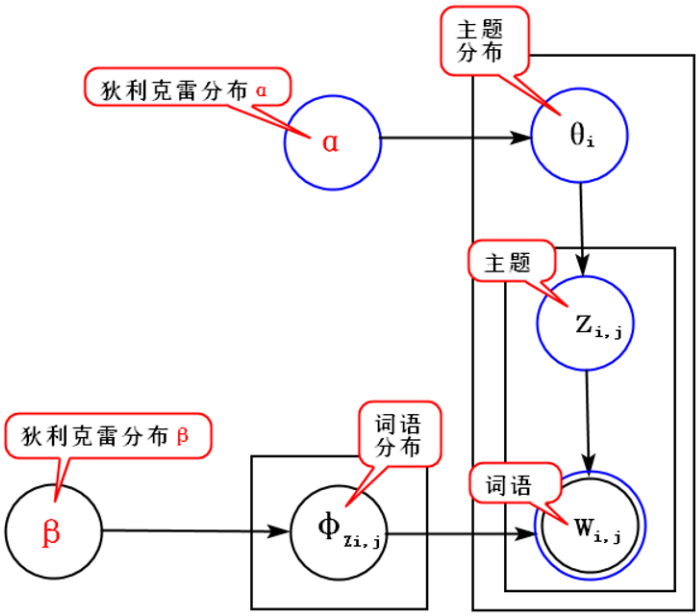

图 6: LDA 主题模型

LDA 的优点包括：

1. 语义分析:
   LDA 可以通过识别文本中的主题，从而实现对文本的语义分析。通过主题模型，我们可以了解文本背后的主要话题和概念。

2. 特征提取：
   LDA 可以将文本数据转化为主题-词分布和文档-主题分布，从而提取文本的关键特征。这些特征可以用于文本分类、聚类和信息检索等任务。

3. 降维：
   LDA 可以将高维的文本数据降低到低维的主题空间。这样可以减少数据的维度，提高后续任务的效率和准确性。

4. 主题发现：
   LDA 可以帮助我们发现文本中的隐藏主题。通过分析主题分布，我们可以发现文本背后的潜在主题结构，揭示文本的深层含义。

4.2 实现细节

1. 数据准备：确保文本数据已经过适当的预处理，包括分词和去除停用词。
2. 模型训练：使用 LDA 模型对文本数据进行训练，以发现潜在的主题结构。
3. 主题提取：分析模型输出，提取出文档集中的主要主题及其相关词汇。
4. 结果可视化：通过词云、主题分布图等形式，直观展示主题模型分析的结果。

这里需要注意的是：

- 由于 LDA 模型是一个文档生成模型，需要不断取样生成主题和词语，并多次重复，建模过程比较复杂，运算时间比较长。因此，根据情况对部分代码加上注释减少它的运行时间；
- 特别需要注意的是，用词袋模型处理数据类型一定是一个列表嵌套。

4.3 模型优化

LDA 主题模型本质上是一个文本聚类模型，并且它与 K-means 聚类算法一样，是需要我们手动输入聚类个数的。

当聚类个数过多，可能出现过度拟合的现象；而聚类个数过少，可能使得不同的主题聚为一类，聚类效果不明显；因此，对于聚类算法来说，确定合适的主题数至关重要。

LDA 主题模型中，对于主题数的选取主要有以下三种：

1. **困惑度（perplexity）：**

   在概率语言模型中, 困惑度是用来评估语言模型优劣的指标, 困惑度一般随着潜在主题数量的增加呈现递减的规律。

   对于训练后的模型构建似然函数并求其最大值，值越大表明提取主题的质量最优，此时的困惑度最小，困惑度的大小与模型质量成反比，较小的困惑度意味着模型对新文本有较好的预测作用。

   在 LDA 主题模型中, 困惑度计算公式如下:

$$\text{Perplexity(D)} = \exp\left\{-\frac{\sum_{d=1}^{M}\log p(w_d)}{\sum_{d=1}^{M}N_d}\right\} \tag{1}$$

   其中, D 表示语料库中的测试集, 共 M 篇文档, Nd 表示每篇文档 d 中的单词数, wd 表示文档 d 中的词, p(wd) 即文档中词 wd 产生的概率。

2. **一致性（coherence）：**

   一种基于主题相关性的定量评价，主要内容是一致性的大小与模型质量成正比。

3. **可视化聚类效果:**

   类似 Excel 中的气泡图，气泡的大小代表主题在文档出现的频率（权重），气泡之间的距离代表主题间的相关性。因此，气泡的间隔越远，说明模型的聚类效果越好。

4.4 运行结果及分析

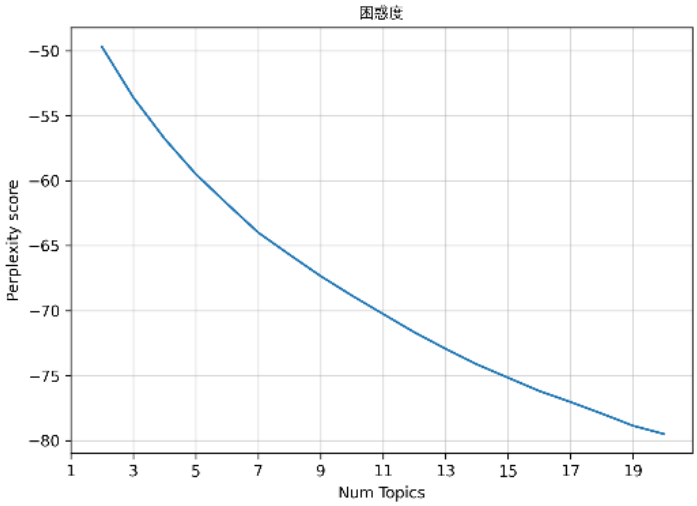

图 7: LDA 模型困惑度

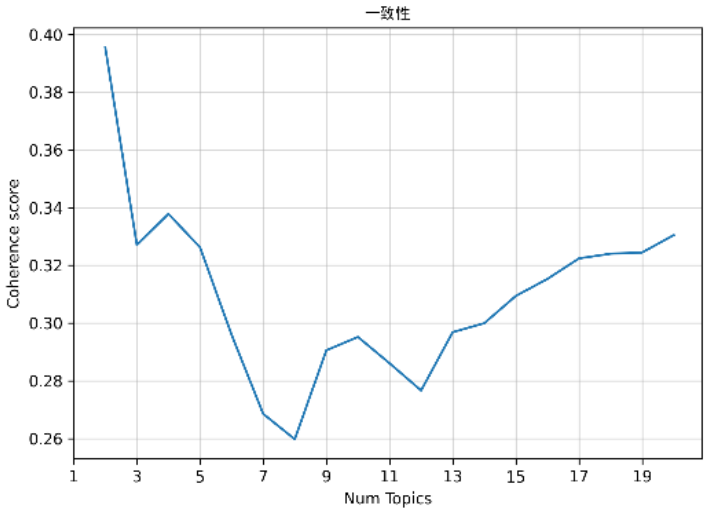

图 8: LDA 模型一致性

困惑度随着话题数量的增加而逐渐降低，表明随着话题数量的增多，模型的预测能力越来越好。一致性则是在话题数量较少的时候有一个较高的起始值，随后迅速下降，然后在话题数量增加到一定程度后又开始上升。综合困惑度与一致性，选择主题数为 4。

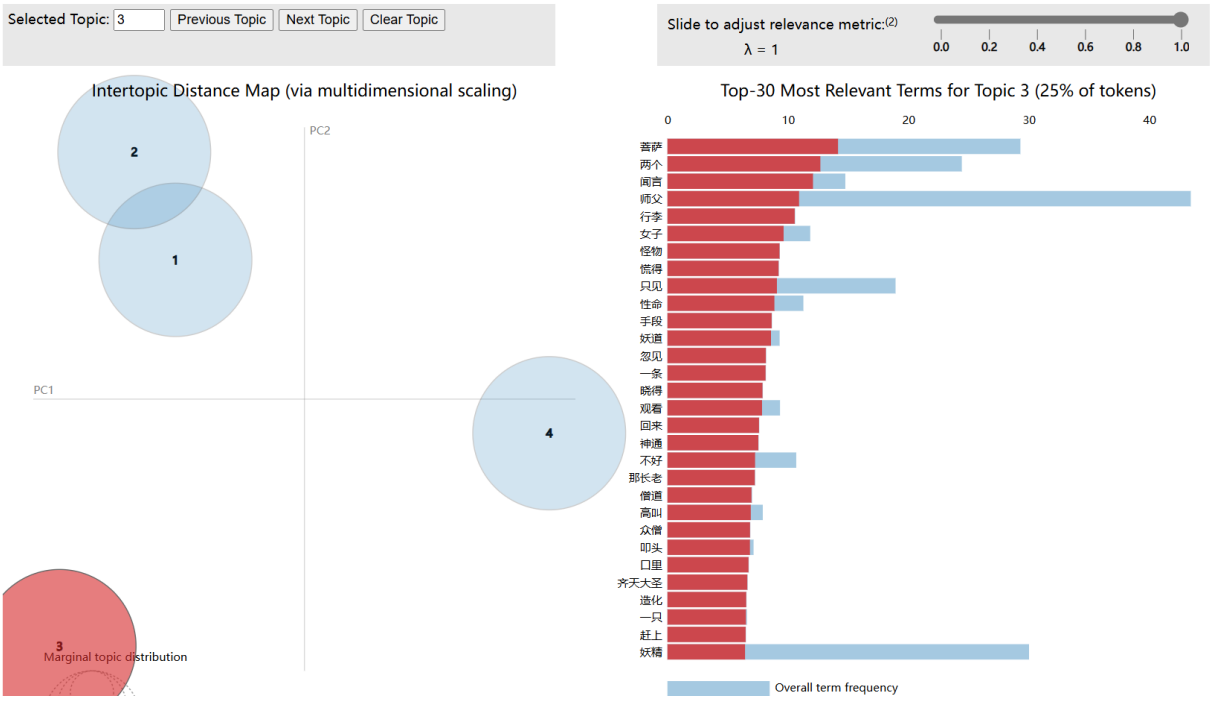

图 9: LDA 模型运行结果

上图为 LDA 模型的可视化界面，左侧的主题距离图使用多维尺度分析展示了不同主题之间的距离关系。右侧的最相关词语列表展示了与主题 3 最相关的 30 个术语，每个词语都有对应的柱状图显示其重要性，红色代表该词语在主题中的频率，蓝色代表其在整体语料库中的频率。

右上角的滑动条可以调节 λ 值，它会改变词语与主题之间的相关性度量，进而影响每个主题下最相关词语的排序和权重。当 λ 值接近 1 时，更倾向于考虑词语在特定主题中的分布；当 λ 值接近 0 时，则更倾向于考虑词语在整体语料库中的分布。当选择的 λ 值为 1 时，最相关词语包括“菩萨”、“师父”、“齐天大圣”等等

结合可视化结果，我们总结出了以下四条主题

- 主题 1：师徒关系及取经情节。
- 主题 2：佛教理念与神佛人物。
- 主题 3：妖怪与战斗场景。
- 主题 4：取经路上的冒险与挑战

5 总结

本文成功地利用 Hadoop 框架实现了小说文本数据的高效处理和分析。通过 Docker 搭建 Hadoop 环境，实现了分布式存储和计算；利用 Jieba 分词工具进行文本预处理，为后续分析提供了基础；基于 MapReduce 框架进行词频统计，并通过词云图进行可视化展示；最后，利用 LDA 主题模型进行主题分析，揭示了小说的潜在主题特征。本研究表明，Hadoop 框架在大规模文本数据处理中具有显著优势，并结合词云和主题模型，可以更深入地理解文本内容的语义和结构。

参考文献

[1] 魏迎. 基于 Hadoop 技术的大规模数据分布式存储技术研究[J]. 自动化与仪器仪表, 2024(10): 182-186. DOI: 10.14016/j.cnki.1001-9227.2024.10.182.

[2] 吴永飞. 基于分布式的多活数据处理平台探索[J]. 金融电子化, 2016(12): 49-50.

[3] 宋杰, 孙宗哲, 毛克明, 等. MapReduce 大数据处理平台与算法研究进展[J]. 软件学报, 2017, 28(03): 514-543. DOI: 10.13328/j.cnki.jos.005169.

[4] 郭蓝天, 李扬, 慕德俊, 等. 一种基于 LDA 主题模型的话题发现方法[J]. 西北工业大学学报, 2016, 34(04): 698-702.

[5] 胡吉明, 陈果. 基于动态 LDA 主题模型的内容主题挖掘与演化[J]. 图书情报工作, 2014, 58(02): 138-142. DOI: 10.13266/j.issn.0252-3116.2014.02.023.

[6] 郝树魁. Hadoop HDFS 和 MapReduce 架构浅析[J]. 邮电设计技术, 2012(07): 37-42.

[7] 李建江, 崔健, 王聃, 等. MapReduce 并行编程模型研究综述[J]. 电子学报, 2011, 39(11): 2635-2642.

[8] 徐戈, 王厚峰. 自然语言处理中主题模型的发展[J]. 计算机学报, 2011, 34(08): 1423-1436.

[9] 邓自立. 云计算中的网络拓扑设计和 Hadoop 平台研究[D]. 中国科学技术大学, 2009.

[10] 朱珠. 基于 Hadoop 的海量数据处理模型研究和应用[D]. 北京邮电大学, 2008.

附录 A　小组分工

表 A.1: 小组分工

姓名	李家泽	张保营	田东旭	贾慧爽
学号	202214870211	202214870213	202214870129	202214870206
分工	部署分布式 Hadoop	词频、词云分析	LDA 主题分析	PPT 制作汇报

附录 B　目录说明

表 A.1: 目录说明

文件	说明
0-docker-hadoop.zip	基于 Docker 的 Hadoop 分布式部署文件
1-Jieba.sh	基于 Python 的 Jieba 命令行分词脚本
2-WordCount.zip	基于 Java 的 MapReduce 词频统计程序
3-WordCloud.R	基于 R 的词云可视化脚本
4-LDA.py	基于 Python 的 LDA 主题分析模型
stopWords.txt	停用词列表
xs.txt	示例小说文本

附录 C　测试方法

运行环境

Docker20.10.22、Java1.8、Hadoop3.3.6、Python3.10、R4.3.3

Hadoop 部署

0-docker-hadoop.zip 压缩包中不含 hadoop 与 jdk 安装包，需自行下载并放到 Dockerfile 同级目录。部署流程如下

1. 修改 Dockerfile，保证安装包文件名一致。

2. 从 Dockerfile 开始构建 hadoop 基础镜像。

   docker build -t hadoop .

3. 执行 docker-compose 启动 3 个容器。

   docker-compose up -d

4. 启动 hadoop 集群和测试验证。

   注：使用 docker ps 查看 master 容器的名字 (假设为 hadoop-master)。

   - 进入 master 容器 (docker exec -it hadoop-master bash)
   - 格式化 HDFS(hdfs namenode -format)
   - 启动 hadoop 集群 (start-all.sh)
   - 查看各个容器的进程 (jps)，如下图所示

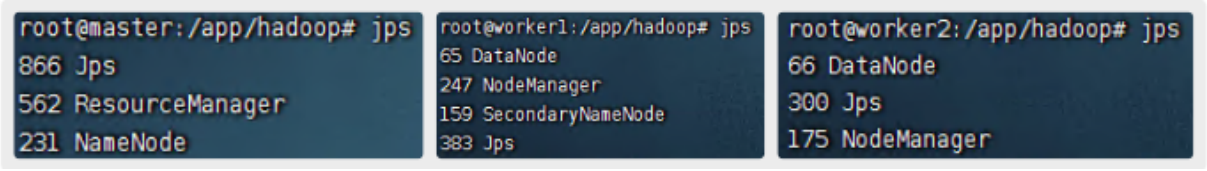

图 C.1: Hadoop 分布式进程

词频分析

1. 运行 sh 文件，使用 Python 的 Jieba 模块进行中文分词
2. MapReduce 词频统计程序源代码编译方法：使用 IDEA 打开工程，等待 maven 将依赖项加载完成，使用 mvn clean install 的方式生成可执行的 jar 包即可进行测试
3. 将编译好的 Jar 文件上传至 Hadoop 运行

   hadoop jar /path/to/WordCount.jar com.tzjava.WordCountDriver /input /output

   - /path/to/WordCount.jar 是 linux 是存储 jar 文件的路径
   - com.tzjava.WordCountDriver 指定运行类名
   - /input /output 是 HDFS 上的输入输出路径
4. 词云图生成：将词频结果与.R 脚本文件放于同一级目录下，直接运行

主题分析

1. 使用 pycharm 打开工程，等待扫描依赖库完成，点击运行即可测试。
2. 如果要进行最优主题选取，需要将注释的代码删除注释，才可进行。
3. 在运行完最优主题选取后，即可把注释恢复，代码会更快地加载完毕。
4. 同时注意在保存最优主题数时需要手动输入，否则代码将进入暂停状态，不会继续运行。

附录 D　关键代码展示

词频统计

Listing 1: **WordCountMapper.java**

```java
package com.tzjava;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.util.regex.Pattern;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private Text outK = new Text();
    private IntWritable outV = new IntWritable(1);
    private static final String STOP_WORDS_PATH = "src/main/resources/stopWords.txt";
    private Set<String> stopWordsSet;
    // 预编译正则
    private static final Pattern WORD_PATTERN = Pattern.compile(".*[a-zA-Z0-9].*");

    private Set<String> loadStopWords(String filePath) throws IOException {
        Set<String> stopWords = new HashSet<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
```

```
26              String line;
27              while ((line = reader.readLine()) != null) {
28                  String[] words = line.split(" ");
29                  for (String word : words) {
30                      stopWords.add(word);
31                  }
32              }
33          }
34          return stopWords;
35      }
36
37      @Override
38      protected void setup(Context context) throws IOException, InterruptedException {
39          try {
40              stopWordsSet = loadStopWords(STOP_WORDS_PATH);
41          } catch (IOException e) {
42              // 处理文件读取失败的情况
43              System.err.println("Failed to load stop words: " + e.getMessage());
44              stopWordsSet = new HashSet<>(); // 初始化为空集合，避免后续操作出错
45          }
46      }
47
48      @Override
49      protected void map(LongWritable key, Text value, Context context) throws IOException,
            InterruptedException {
50          // 获取一行
51          String line = value.toString();
52          // 切割
53          String[] words = line.split(" ");
54          // 循环写出
55          for (String word : words) {
56              // 去除单字、标点符号
57              if (word.length() <= 1)
58                  continue;
59              // 使用正则判断是否为字母、数字等，如果为是，则continue
60              if (WORD_PATTERN.matcher(word).matches())
61                  continue;
62              // 剔除停用词，停用词存储在stopWords.txt中
63              if (stopWordsSet.contains(word))
64                  continue;
65              // 封装outK
66              outK.set(word);
```

```
67              // 写出
68              context.write(outK, outV);
69          }
70      }
71  }
```

Listing 2: **WordCountReducer.java**

```
1   package com.tzjava;
2
3   import org.apache.hadoop.io.IntWritable;
4   import org.apache.hadoop.io.Text;
5   import org.apache.hadoop.mapreduce.Reducer;
6
7   import java.io.IOException;
8
9   public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
10      private IntWritable outV = new IntWritable();
11
12      @Override
13      protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
            IOException, InterruptedException {
14          int sum = 0;
15          // text,(1,1)
16          // 累加
17          for (IntWritable value : values) {
18              sum += value.get();
19          }
20          outV.set(sum);
21          // 写出
22          context.write(key, outV);
23      }
24  }
```

LDA 主题分析

Listing 3: **LDA.py**

```
1   # coding='utf-8'
2   import os
3   import datetime
4   import warnings
```

```
import pandas as pd
from gensim.models import LdaModel
from gensim.corpora import Dictionary
from gensim import corpora, similarities, models
from gensim.models.coherencemodel import CoherenceModel
import matplotlib.pyplot as plt
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
warnings.filterwarnings("ignore")

# 读取分词好的文本文件
def infile(filepath,wordstop):
    train = []
    with open(filepath, 'r', encoding='utf8') as fp:
        for line in fp:
            new_line = []
            if len(line) > 1:
                line = line.strip().split(' ')
                for w in line:
                    if len(w) > 1 and w not in wordstop:
                        new_line.append(w)
            if len(new_line) > 1:
                train.append(new_line)
    return train

# 处理文本数据
def deal(train):
    id2word = corpora.Dictionary(train)      # 创建词典
    texts = train                            # 创建语料库
    corpus = [id2word.doc2bow(text) for text in texts]   # 词频矩阵

    # 使用 TF-IDF (词频-逆文档频率模型)
    tfidf = models.TfidfModel(corpus)
    corpus = tfidf[corpus]

    # 保存词典和语料库
    os.makedirs('tmp', exist_ok=True)
    id2word.save('tmp/deerwester.dict')
    corpora.MmCorpus.serialize('tmp/deerwester.mm', corpus)

    return id2word, texts, corpus

```

```
47 # 运行标准 LDA 模型
48 def run(corpus_1, id2word_1, num, texts):
49     lda_model = LdaModel(corpus=corpus_1,
50                          id2word=id2word_1,
51                          num_topics=num,
52                          passes=60,
53                          alpha=(50 / num),
54                          eta=0.01,
55                          random_state=42)
56
57     # 困惑度
58     perplex = lda_model.log_perplexity(corpus_1)
59
60     # 一致性
61     coherence_model_lda = CoherenceModel(model=lda_model, texts=texts, dictionary=id2word_1,
           coherence='c_v')
62     coherence_lda = coherence_model_lda.get_coherence()
63
64     return lda_model, coherence_lda, perplex
65
66 # 保存 LDA 可视化结果
67 def save_visual(lda, corpus, id2word, name):
68     d = gensimvis.prepare(lda, corpus, id2word)
69     pyLDAvis.save_html(d, name + '.html')
70
71 if __name__ == '__main__':
72     # 文本名
73     stopwords =[line.strip() for line in open('stopWords.txt',encoding='utf-8').readlines()]
74     train = infile('西游记-分词.txt',stopwords)
75     # 处理数据
76     id2word, texts, corpus = deal(train)
77
78     ## 训练标准 LDA 模型
79     # perplexity = []              #困惑度数组
80     # coherence_values = []        #一致性数组
81     # model_list = []
82     # for num_topics in range(2, 21, 1):
83     #     lda, coherence_lda, perplex = run(corpus, id2word,num_topics, texts)
84     #     model_list.append(lda)
85     #     perplexity.append(perplex)
86     #     coherence_values.append(coherence_lda)
87     #
```

```
    ##绘制Perplexity - Coherence - Topic折线图
    # plt.figure(figsize=(16, 5), dpi=200)
    # x = range(2, 21, 1)
    # ax1 = plt.subplot(1, 2, 1)
    # plt.plot(x, perplexity)
    # plt.xlabel("Num Topics")
    # plt.ylabel("Perplexity score")
    # plt.xticks(range(1, 21, 2)) # 设置刻度
    # plt.title('困惑度',fontproperties="SimHei")
    # plt.grid(True, alpha=0.5)
    # ax2 = plt.subplot(1, 2, 2)
    # plt.plot(x, coherence_values)
    # plt.xlabel("Num Topics")
    # plt.ylabel("Coherence score")
    # plt.xticks(range(1, 21, 2)) # 设置刻度
    # plt.title('一致性',fontproperties="SimHei")
    # plt.grid(True, alpha=0.5)
    # plt.savefig('./困惑度与一致性.png', dpi=300)

    now = datetime.datetime.now()
    current_time = now.time()
    nicetopicnum = input(f"当前时间:{current_time} " + "请输入最佳主题数:")
    print(f"最佳主题数为{nicetopicnum}")

    lda, coherence_lda, perplex = run(corpus, id2word, int(nicetopicnum), texts)
    topic_list = lda.print_topics()

    #保存主题列表
    with open('xyj_topics.txt', 'w', encoding='utf-8') as f:
        for t in topic_list:
            f.write(' '.join( str(s) for s in t) + '\n')

    #保存LDA可视化结果
    save_visual(lda, corpus, id2word, 'xyj')
```