

**计算机与网络空间安全学院学生实验报告**

**实验课程名称：** 大数据导论 **教师： 林鑫泓 \_\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | **大数据软件应用与技术实践** | | | **实验成绩** |  |
| **学生姓名** | **徐松福**  **叶继禹**  **范嘉明** | **学 号** | **121052021023**  **121052021010**  **121052021038** | **年级专业班级** | **21级软工一班** |
| **小组成员** | **徐松福、叶继禹、范嘉明** | | | **实验日期** | **2022年 12 月** |

# 1 实验目的和要求

## 1.1 实验目的

① 实现一个基于协同过滤的电影推荐系统，通过实践学习如何处理和分析大规模数据集。

② 掌握使用Hadoop和MapReduce进行大数据集的存储、管理和处理。

③ 应用数据挖掘技术，特别是协同过滤算法，来预测和推荐用户可能喜欢的电影。

④ 通过实验，熟悉数据预处理、数据统计分析以及机器学习模型的开发和评估。

## 1.2 实验软硬件环境

① 操作系统：Linux CentOS 7

② Java：open-jdk-1.8

③ Apache Hadoop：3.3.6

④ zookeeper：3.8.3

⑤ HBase版本：2.5.6

⑥ MapReduce

⑦ Yarn

⑧ mahout：0.13.0

## 1.3 实验要求

1. 结合Netflix Dataset或ml-25m，按照实验步骤完成实验。

2. Netflix数据集包含了1999.12.31至2005.12.31期间由网站用户提供的超过一亿条电影评价。Netflix Dataset.7z压缩文件包含电影信息、training set（训练集）、probe set（探测集）和qualifying set(评估集)。压缩文件的详细信息如下图1所示：

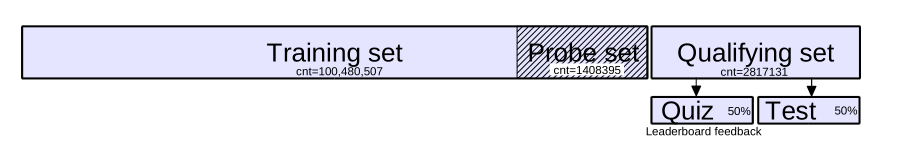


图1 Netflix Dataset说明

3. 实验成绩按照各步骤完成情况给分。

4. 大数据软件需包括教材当中有的，可以是介绍过的也可以是没介绍过的，尽可能地使用本学期学习并实验过的大数据软件和方法。

# 2、实验步骤

1. 独立设计一套切实可行的数据挖掘任务task，并简要介绍该任务的各个步骤step，使用的关键技术与软件应用。（10分）

2. 详细介绍数据准备过程，例如：如何对数据进行预处理，如何做数据持久化存储。（10分）

3. 详细介绍数据统计分析的过程，例如：统计、分析电影评价的数量、用户的数量、电影的数量等，统计、分析电影评价的平均值、方差等。（10分）

4. 详细介绍数据挖掘的算法、过程和结果。例如：使用了某个数据挖掘的算法，或编辑了某段代码，接着对多少用户和多少电影进行了协同过滤，得到了什么结果或什么成果，有什么结论。（10分）

# 3实验记录

## 独立设计一套切实可行的数据挖掘任务task，并简要介绍该任务的各个步骤step，使用的关键技术与软件应用。（10分）

**设计数据挖掘任务**

**目标：**

构建协同过滤电影推荐系统。

**关键步骤：**

**1.数据预处理：**

**①数据探索：**首先对数据进行探索性分析，了解数据的分布、类型和潜在问题。

**②清洗：**处理缺失值、重复记录和异常值。例如，对于缺失的电影评分，可以选择忽略这些记录或使用平均评分进行填充。

**③格式化与转换：**确保所有数据采用统一和适合的格式，如统一日期格式，对分类数据进行编码。

**2.数据持久化存储：**

**①HDFS应用：**在Hadoop HDFS上存储原始数据和预处理后的数据，设计合理的文件结构以优化读写效率。

**②HBase设计：**在HBase中创建适当的表来存储用户信息、电影信息和评分数据，设计有效的Row Key策略。

**3.数据分析：**

**①统计分析：**计算关键指标，如电影的平均评分、用户活跃度等。

**②趋势分析：**分析评分随时间的变化趋势，识别热门电影和活跃用户。

**4.推荐算法实现：**

**①算法选择：**确定使用基于用户的还是基于物品的协同过滤，或者尝试混合推荐方法。

**②特征工程：**选择合适的特征，如用户评分、评分次数、电影类别等。

**③模型开发：**在选择的框架（如Mahout或自定义MapReduce程序）中实现推荐模型。

**5.系统评估：**

**①评估指标：**确定评估指标，如准确度、覆盖率、RMSE等。

**②交叉验证：**应用交叉验证方法来评估模型的泛化能力。

**关键技术与软件：**

**①Hadoop HDFS：**用于存储和处理大规模数据集。

**②HBase：**用于管理结构化数据，如用户资料、电影信息和评分记录。

**③MapReduce：**用于处理大规模数据集的统计和分析。

**④Mahout/自定义算法：**实现协同过滤算法，选择开源的Mahout库或开发自定义算法。

**⑤附加工具（可选）：**使用如Apache Spark进行更快速的数据处理，或使用Python等其他语言进行数据探索和可视化。

## 详细介绍数据准备过程，例如：如何对数据进行预处理，如何做数据持久化存储。（10分）

**数据清洗：**

①过滤非法的数据格式，如不正确的用户ID、电影ID或评分数据。

②确保所有数据项都是有效的数字，并且评分在合理的范围内（1到5）。

**数据转换：**

①将原始的数据行转换成（电影ID，用户ID:评分）的格式，为后续的处理统一了数据格式。

②这样的输出格式便于进行聚合操作，也为构建用户-电影评分矩阵奠定了基础。

**数据格式化：**

①将清洗后的数据输出为易于阅读和处理的格式，即每行包含一个电影ID和对应的用户评分列表。

②这种格式化的数据便于后续分析每部电影的评价数量、平均评分、方差等。

**步骤1：数据上传到HDFS**

**1.准备数据文件**

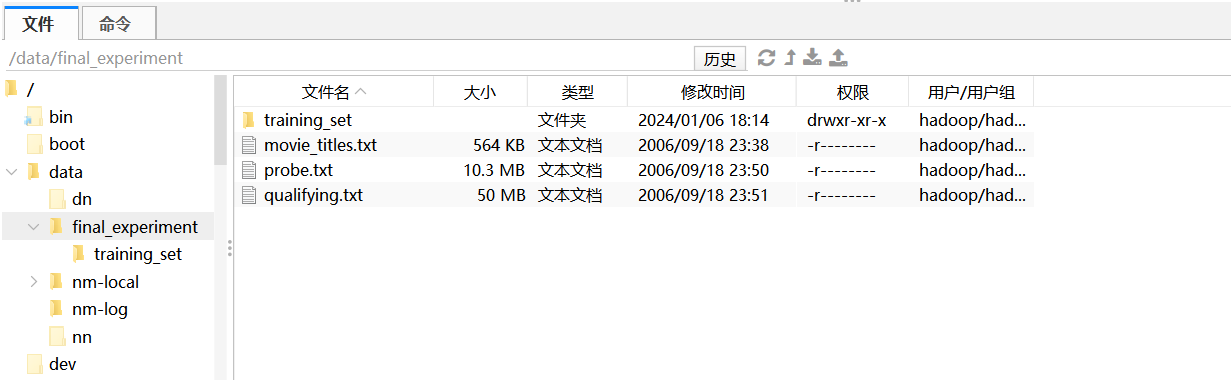
①电影信息 (movie\_titles.txt): 包含电影ID、发行年份和电影标题。

②预测集 评分数据 (mv\_XXXXXXX.txt): 多个文件，每个文件包含一部电影的所有评分，格式为：用户ID,评分,日期。

③探测集 (probe.txt): 包含用于模型验证的用户ID和电影ID。

④资格验证集 (qualifying.txt): 用于模型最终测试的用户ID和电影ID。

**2.将数据文件上传到虚拟机上**



1. **数据上传到HDFS**

1. 创建HDFS目录

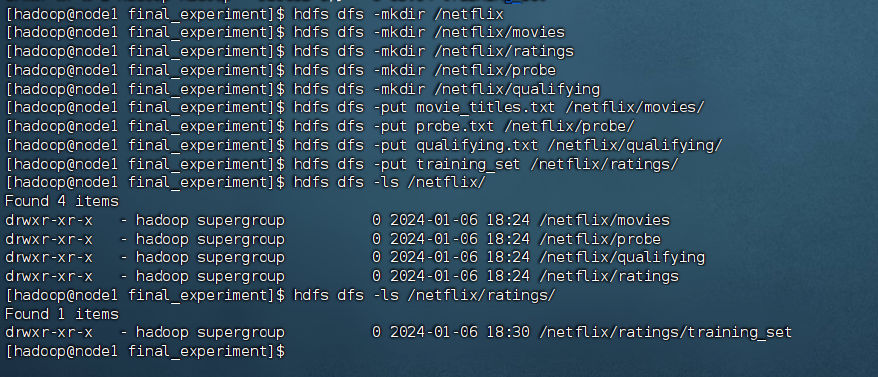
|  |
| --- |
| hdfs dfs -mkdir /netflix  hdfs dfs -mkdir /netflix/movies  hdfs dfs -mkdir /netflix/ratings  hdfs dfs -mkdir /netflix/probe  hdfs dfs -mkdir /netflix/qualifying |

1. 上传文件到HDFS

|  |
| --- |
| hdfs dfs -put movie\_titles.txt /netflix/movies/  hdfs dfs -put probe.txt /netflix/probe/  hdfs dfs -put qualifying.txt /netflix/qualifying/  hdfs dfs -put training\_set /netflix/ratings/ |

1. 验证上传

|  |
| --- |
| hdfs dfs -ls /netflix/  hdfs dfs -ls /netflix/ratings/ |



**步骤2：数据预处理**

**Mapper类 (RatingMapper.java)**

Mapper 类的主要功能是解析每行输入的文本数据（在这个情境下是用户对电影的评分数据），执行初步的数据清洗工作，然后输出处理后的结果。具体来说，它做了以下事情：

**解析：**将输入文本按逗号分割，提取用户ID、电影ID、评分和评分日期。

**数据验证：**检查提取的电影ID和用户ID是否有效（是否为数字），以及评分是否在合法范围内（1到5之间）。

**输出：**对于有效的记录，Mapper 输出电影ID作为key，用户ID和评分的组合作为value。

|  |
| --- |
| import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Mapper;  import java.io.IOException;  public class RatingMapper extends Mapper<LongWritable, Text, Text, Text> {    // Map方法会为每个输入的key/value对调用一次  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  // 将输入的value转换成String  String line = value.toString();    // 分割数据行为单独的元素  String[] elements = line.split(",");    // 期望的数据格式是：UserID,MovieID,Rating,Date  if (elements.length == 4) {  String userID = elements[0].trim();  String movieID = elements[1].trim();  String rating = elements[2].trim();  String date = elements[3].trim();    // 简单的数据清洗：检查电影ID和用户ID是否为数字，评分是否在1到5之间  if (isValidNumber(userID) && isValidNumber(movieID) && isValidRating(rating)) {  // 输出key为电影ID，value为用户ID和评分的组合，以便于后续的Reducer处理  context.write(new Text(movieID), new Text(userID + ":" + rating));  }  }  }    // 辅助函数：检查字符串是否能转换为数字  private boolean isValidNumber(String str) {  try {  Integer.parseInt(str);  return true;  } catch (NumberFormatException e) {  return false;  }  }    // 辅助函数：检查评分是否在1到5之间  private boolean isValidRating(String rating) {  try {  int rate = Integer.parseInt(rating);  return rate >= 1 && rate <= 5;  } catch (NumberFormatException e) {  return false;  }  }  } |

**Reducer类 (RatingReducer.java)**

Reducer 类接收Mapper的输出，并对每个电影ID（key）对应的所有评分（values列表）进行处理。在这个简单的例子中，Reducer的作用不多：

**合并评分：**将同一电影ID的所有评分合并到一个字符串中。

**输出：**将电影ID和合并后的评分字符串输出。

|  |
| --- |
| import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Reducer;  import java.io.IOException;  public class RatingReducer extends Reducer<Text, Text, Text, Text> {  // Reduce方法会为每个输入的key和相关的value列表调用一次  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {  // 对于每个电影ID，我们将所有的用户评分合并成一个字符串  StringBuilder allRatings = new StringBuilder();    for (Text value : values) {  allRatings.append(value.toString()).append(", ");  }    // 输出key为电影ID，value为所有用户评分的字符串  context.write(key, new Text(allRatings.toString()));  }  } |

**Driver类 (RatingDriver.java)**

Driver 类是整个MapReduce作业的配置和启动中心。它设定了作业的各项参数，并启动作业。具体包括：

**配置作业：**设定作业的名称，指定用到的Mapper和Reducer类，设置输出数据的key和value类型。

**设置输入输出路径：**指定作业的输入数据路径和输出数据路径。

**作业执行：**启动作业，并等待其完成，最后根据作业执行的成功与否退出程序。

|  |
| --- |
| import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class RatingDriver {    public static void main(String[] args) throws Exception {  if (args.length != 2) {  System.err.println("Usage: RatingDriver <input path> <output path>");  System.exit(-1);  }  Configuration conf = new Configuration();  Job job = Job.getInstance(conf, "Rating Analysis");  // 设置MapReduce任务的内存配置  job.getConfiguration().set("mapreduce.map.memory.mb", "512");  job.getConfiguration().set("mapreduce.map.java.opts", "-Xmx512m");  job.getConfiguration().set("mapreduce.reduce.memory.mb", "512");  job.getConfiguration().set("mapreduce.reduce.java.opts", "-Xmx512m");  job.setJarByClass(RatingDriver.class);  job.setMapperClass(RatingMapper.class);  job.setReducerClass(RatingReducer.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(Text.class);  FileInputFormat.addInputPath(job, new Path(args[0]));  FileOutputFormat.setOutputPath(job, new Path(args[1]));  boolean success = job.waitForCompletion(true);  System.exit(success ? 0 : 1);  }  } |

**编译和打包**

将上述代码保存为 .java 文件，然后使用Java编译器编译这些文件，最后将它们打包成一个 .jar 文件。

①创建项目目录：

|  |
| --- |
| mkdir -p ~/mapreduce/netflix  cd ~/mapreduce/netflix |

②创建源代码和类文件目录：

|  |
| --- |
| mkdir src bin lib |

③编写和存储MapReduce代码：

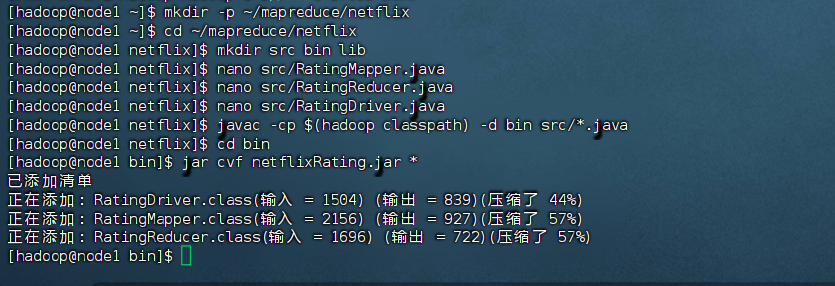
|  |
| --- |
| nano src/RatingMapper.java  nano src/RatingReducer.java  nano src/RatingDriver.java |

④编译代码：

|  |
| --- |
| javac -cp $(hadoop classpath) -d bin src/\*.java |

⑤打包成JAR：

|  |
| --- |
| cd bin  jar cvf netflixRating.jar \* |

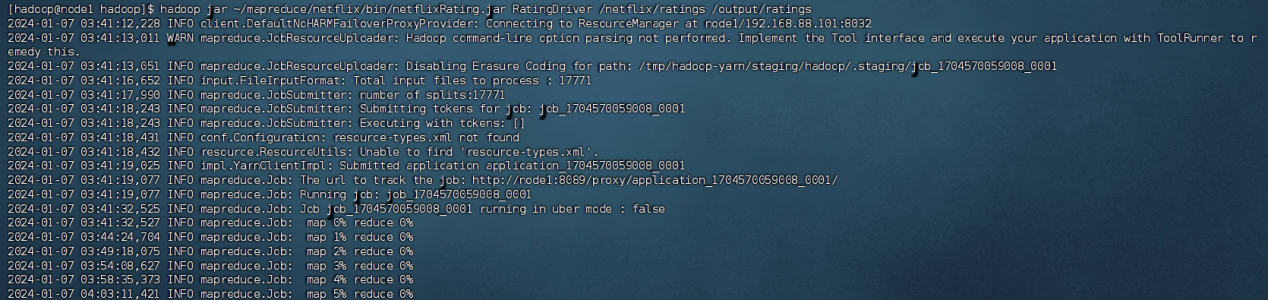


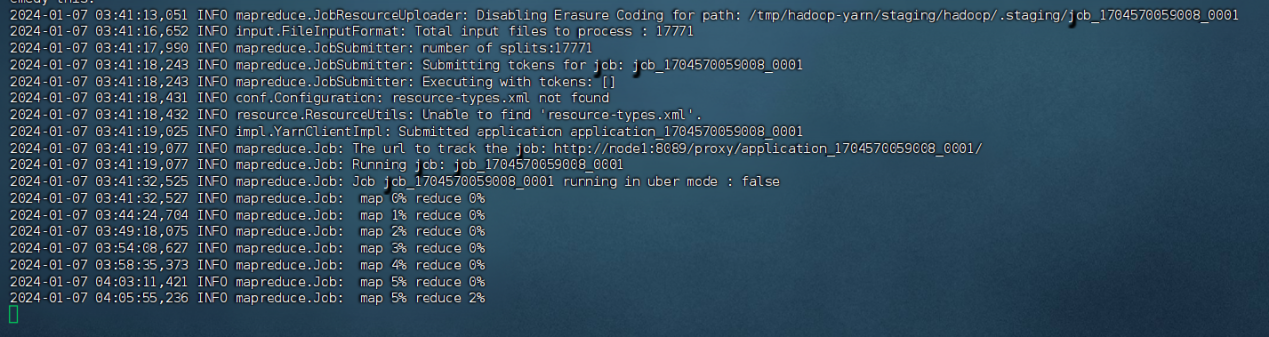
将training\_set目录中的所有文件移动到一个公共的父目录

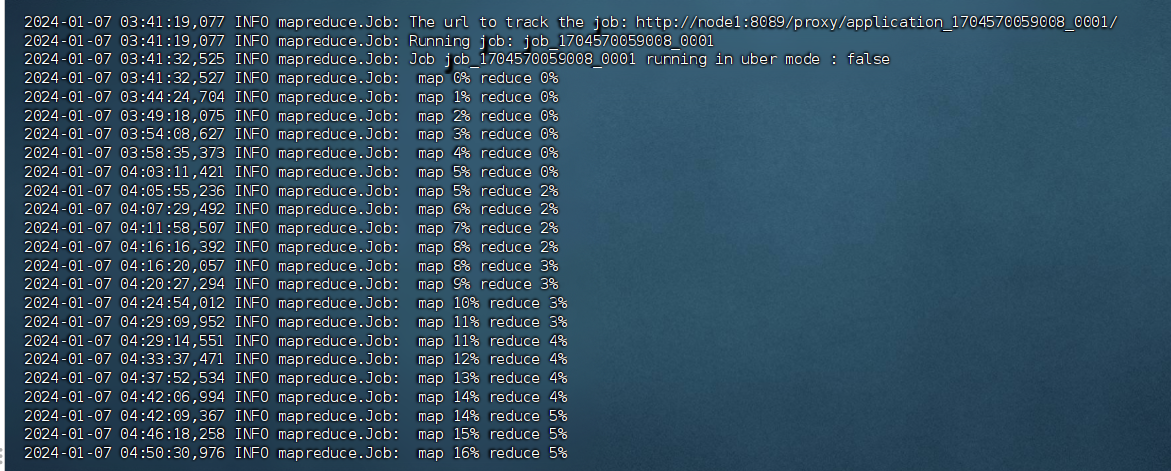
|  |
| --- |
| hdfs dfs -mv /netflix/ratings/training\_set/\* /netflix/ratings/ |

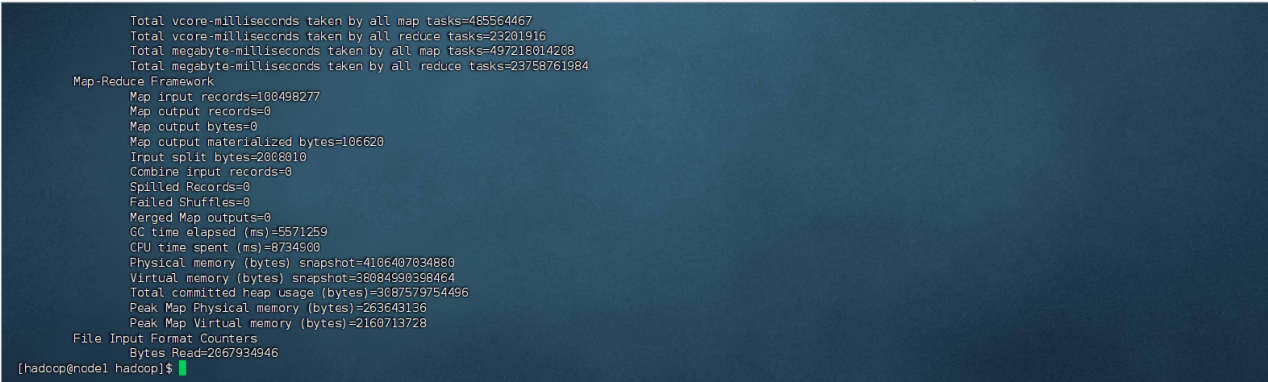
⑥运行MapReduce作业：

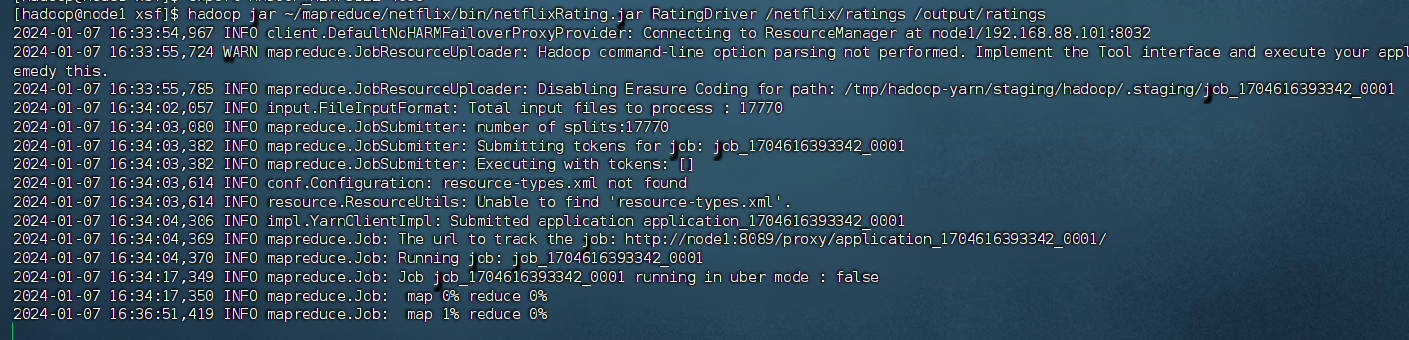
|  |
| --- |
| hadoop jar ~/mapreduce/netflix/bin/netflixRating.jar RatingDriver /netflix/ratings /output/ratings |

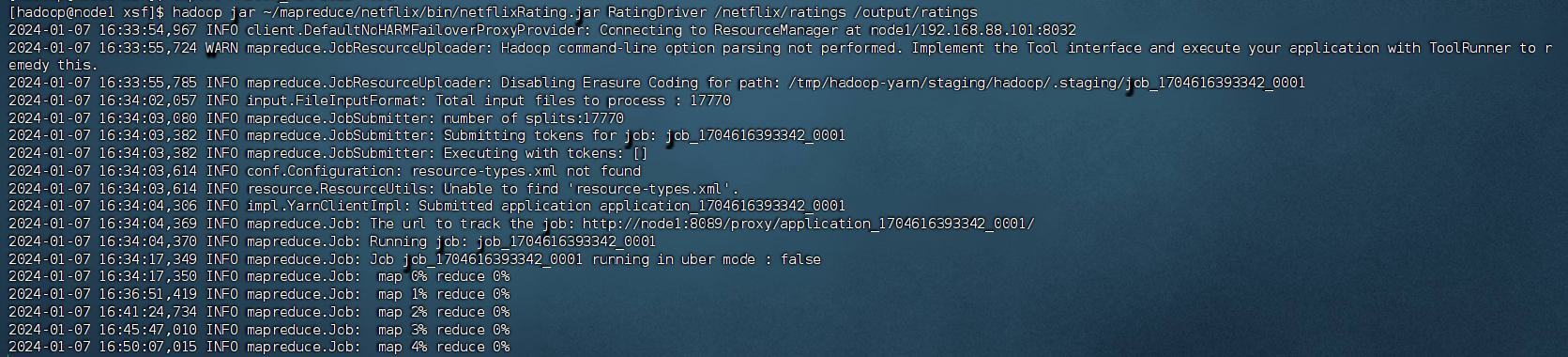




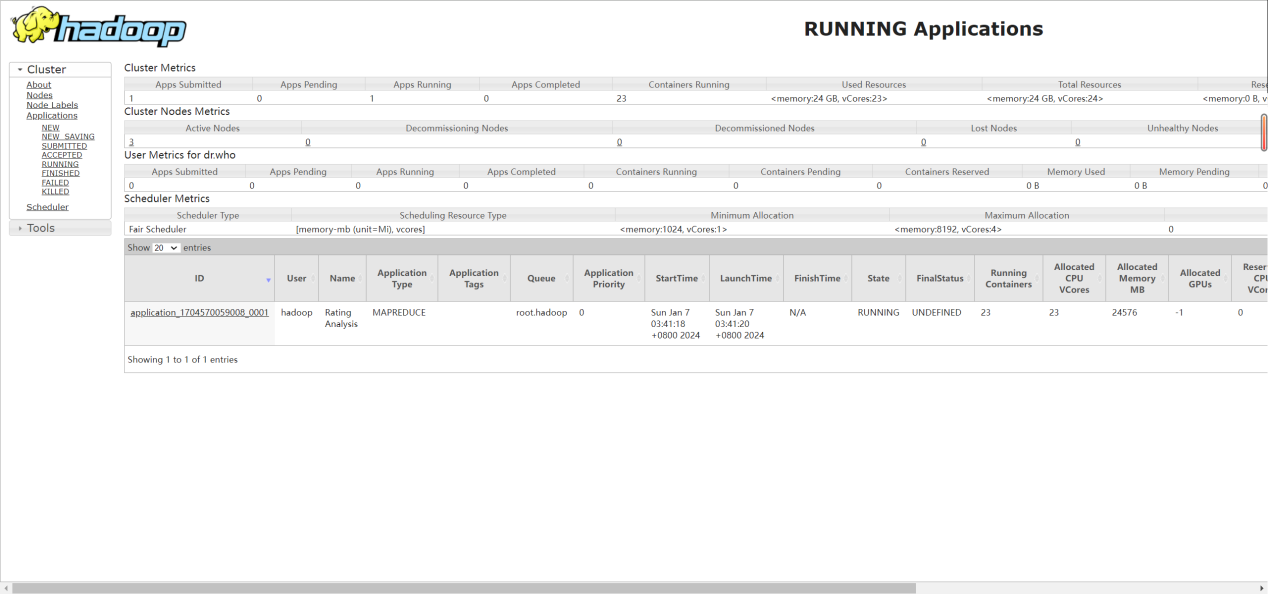


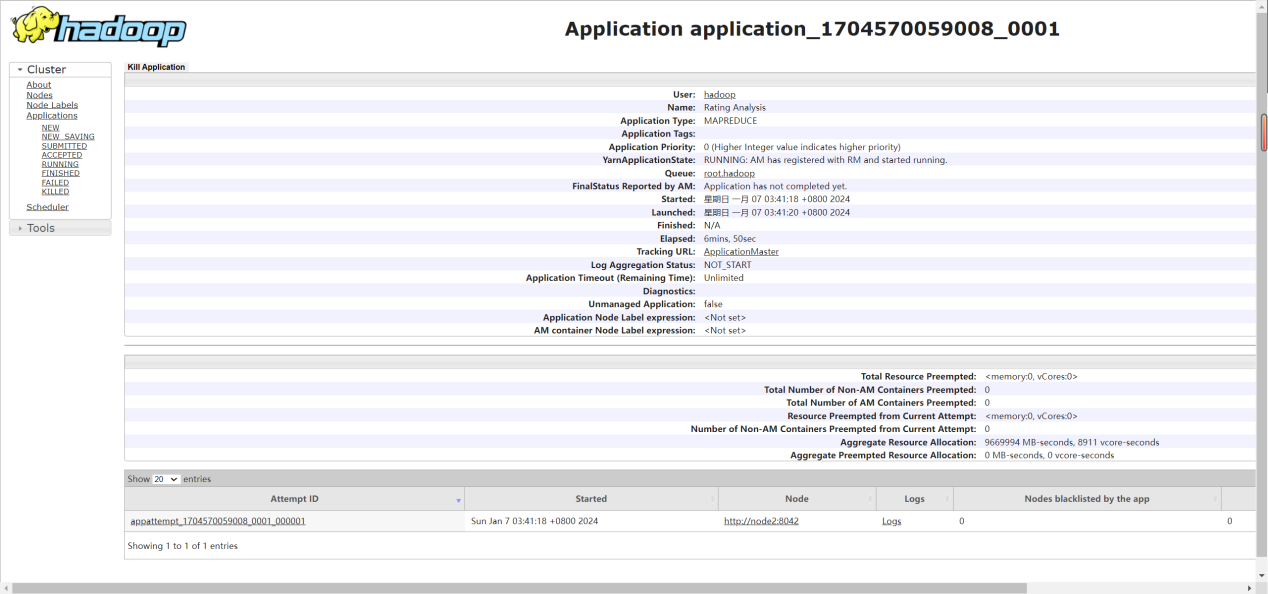






使用web界面来查看MapReduce任务





**数据的持久化存储：**

在Hadoop HDFS上存储原始数据和预处理后的数据，设计合理的文件结构以优化读写效率。

## 3. 详细介绍数据统计分析的过程，例如：统计、分析电影评价的数量、用户的数量、电影的数量等，统计、分析电影评价的平均值、方差等。（10分）

# 4 实验总结