第四次实验报告

姓名	学号	邮箱	院系
姜帅	221220115	js13156223725@163.com	计算机学院

任务1: 社交网络中的互相关注好友

设计思路

- 输入阶段: 读取文件夹中的所有文本文件，逐行解析用户及其关注列表。
- Map阶段: 为每对用户关系生成键值对，键为排序后的用户对，值表示关注发起者, 例如: key为`A-B`, value为`A`
- Reduce阶段: 收集所有的值, 得到每个值对应的关注发起者的列表, 检查列表长度是否为**2**，且两个值是否不同。若满足，说明两个用户互相关注。输出互相关注的用户对，值为空。

核心代码

以下是Mapper的核心代码片段, 附注释

```
// 输入格式: user_name: friend1 friend2 friend3 ...
String line = value.toString().trim();
if (line.isEmpty() || !line.contains(":")) {
    return;
}
// 分割用户名和朋友列表
String[] parts = line.split(":");
String user = parts[0].trim();
String[] friends = parts.length > 1 ? parts[1].trim().split("\\s+") :
new String[0];
// 为每个朋友生成键值对
for (String friend : friends) {
    if (!friend.isEmpty()) {
        // 创建排序后的用户对作为key，确保user_A < user_B
        String pairKey = user.compareTo(friend) < 0 ? user + "-" +
friend : friend + "-" + user;
        // value为当前用户，表示这个用户关注了这个朋友
        context.write(new Text(pairKey), new Text(user));
    }
}
```

Reducer的核心代码片段:

```java
// 收集所有用户
List<String> users = new ArrayList<>();
for (Text value : values) {
    users.add(value.toString());
}
// 如果恰好有两个不同的用户，说明是互相关注
if (users.size() == 2 && !users.get(0).equals(users.get(1))) {
    context.write(key, new Text(""));
}
```

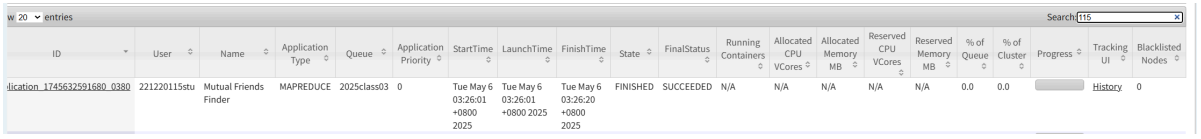
执行过程及结果

在集群上输入以下命令执行作业:

```
yarn jar /home/221220115stu/task1-1.0-SNAPSHOT.jar
com.example.MutualFriends /user/root/Exp4 /user/221220115stu/lab4/task1
```

输出结果文件位于`/user/221220115stu/lab4/task1`下, 输出结果的部分截图如下:

执行报告



任务2: 社交网络寻找共同关注

设计思路

本任务旨在通过MapReduce框架查找社交平台中用户对的共同关注者。输入包括两部分：第一部分为互相关注用户对文件（格式为`userA-userB`），第二部分为包含用户关注列表的文件夹（格式为`user_name: friend1 friend2 ...`）。任务目标是计算每对互相关注用户的共同关注者，并根据共同关注者数量与阈值`x`的关系，将结果分为两类输出：`small`（共同关注者数量

≤x）和`large`（共同关注者数量>x）。整体流程如下：

该任务采用两阶段mapreduce处理

- 第一阶段mapper, 读入用户关注列表和互相关注对**两个文件**, 使用一个mapper统一处理, 处理过程为:
  - 如果输入来自用户关注列表, 例如`A: f1 f2 f3`, 则输出的key为`A`, $value$为`L: f1,f2,f3` 这里的**L**起到一个标签的作用, 具体作用在下一条说明
  - 如果输入来自互相关注对, 例如`A-B`, 则分别输出`(A, P:B)`和`(B, P:A)`, 这里的**P**同样是起到标签的作用, 目的是在接下来的reduce阶段中将用户关注列表信息和互相关注信息区分开
- 第一阶段的reducer处理过程为:
  - 对每个user, 收集其完整的关注列表 (L开头的 value)
  - 对每个user, 收集其完整的与之互相关注的用户 (P开头的value)
  - 对于每一对`user - P: other_user`, 输出`(user-other_User, user 的关注列表)`, 用于下一阶段配对交集计算
- 第二阶段mapper:
  - 无实质性作用, 只是起到一个传递作用
- 第二阶段reducer:
  - 对每个互相关注对`user1-user2`, 收集user1和user2的互相关注列表, 存放于两个HashSet中, 计算这两个集合的交集
  - 判断交集大小, 即user1和user2共同关注者的数量, 与阈值x的关系, 按照与x的大小关系判断是输出到large文件中还是small文件中

核心代码

一阶段Mapper的核心代码:

```
String line = value.toString().trim();
if (line.contains(":")) { // 处理关注列表: user: friend1 friend2 ...
    String[] parts = line.split(":");
    if (parts.length != 2) return;
    String user = parts[0].trim();
    String[] friends = parts[1].trim().split("\\s+");
    context.write(new Text(user), new Text("L:" + String.join(",",
friends)));
} else if (line.contains("-")) { // 处理互相关注对: userA-userB
    String[] users = line.split("-");
    if (users.length != 2) return;
    String userA = users[0].trim();
    String userB = users[1].trim();
    context.write(new Text(userA), new Text("P:" + userB));
    context.write(new Text(userB), new Text("P:" + userA));
}
```

一阶段Reducer的核心代码:

```
List<String> partners = new ArrayList<>();
String friendList = "";
// 收集互相关注信息 P:... 和关注列表信息 L:...
for (Text val : values) {
    String value = val.toString();
    if (value.startsWith("P:")) {
        partners.add(value.substring(2));
    } else if (value.startsWith("L:")) {
        friendList = value.substring(2);
    }
}
// 对于每一对user-P:other_user构建输出
for (String partner : partners) {
    String pair = key.toString().compareTo(partner) < 0 ? key + "-" +
partner : partner + "-" + key;
    context.write(new Text(pair), new Text(friendList));
}
```

二阶段Mapper的核心代码

```
String[] parts = value.toString().split("\\t");
if (parts.length != 2) return;
context.write(new Text(parts[0]), new Text(parts[1]));
```

二阶段Reducer的核心代码

```
List<String> lists = new ArrayList<>();
// 对于user1-user2, 收集user1和user2的关注列表信息
for (Text val : values) {
    lists.add(val.toString());
}
if (lists.size() != 2) return;
// 将user1和user2的关注列表转化为集合, 取交集
Set<String> set1 = new HashSet<>
(Arrays.asList(lists.get(0).split(",")));
Set<String> set2 = new HashSet<>
(Arrays.asList(lists.get(1).split(",")));
set1.retainAll(set2);

if (set1.isEmpty()) return;
// 构建目标格式的输出
String commonFriends = String.join(" ", set1);
Text formattedLine = new Text(key.toString() + ": " + commonFriends);
// 按照阈值,将输出结果分流
if (set1.size() <= threshold) {
    mos.write("small", NullWritable.get(), formattedLine,
"small/part");
} else {
    mos.write("large", NullWritable.get(), formattedLine,
"large/part");
```

```
21  }
```

执行过程及结果

在集群上输入以下命令执行作业: (x以10为例)

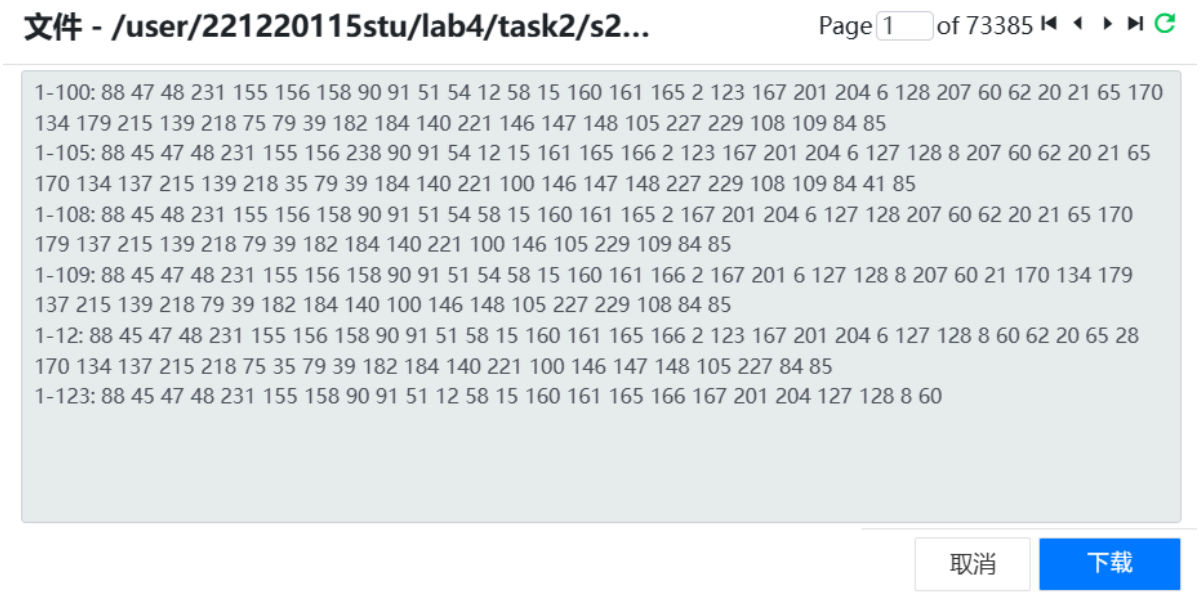
```
yarn jar /home/221220115stu/task1-1.0-SNAPSHOT.jar
com.example.CommonFollowers /user/221220115stu/lab4/task1/part-r-00000
/user/root/Exp4 10 /user/221220115stu/lab4/task2/s1
/user/221220115stu/lab4/task2/s2
```

输出结果文件位于`/user/221220115stu/lab4/task2/s2下` 分为`/user/221220115stu/lab4/task2/s2/small/part-r-00000`和`/user/221220115stu/lab4/task2/s2/large/part-r-00000`

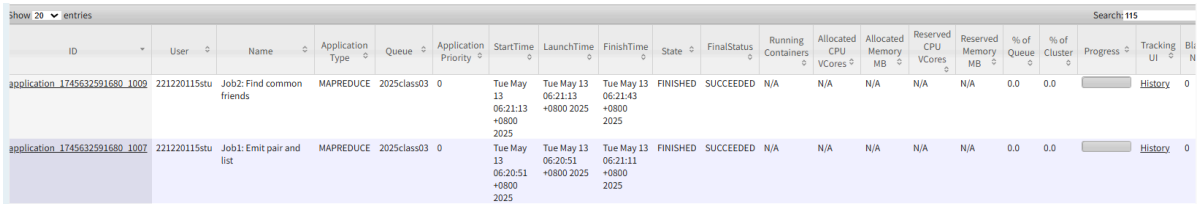
输出结果的部分截图如下:

- small:

- large:



执行报告



ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Bl N
application_1745632591680_1009	221220115stu	Job2: Find common friends	MAPREDUCE	2025class03	0	Tue May 13 06:21:13 +0800 2025	Tue May 13 06:21:13 +0800 2025	Tue May 13 06:21:43 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1745632591680_1007	221220115stu	Job1: Emit pair and list	MAPREDUCE	2025class03	0	Tue May 13 06:20:51 +0800 2025	Tue May 13 06:20:51 +0800 2025	Tue May 13 06:21:11 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

任务3: 好友推荐

设计思路

本任务通过MapReduce框架为社交平台用户推荐好友. 输入包括互相关注用户对文件（格式为`userA-userB`）和用户关注列表文件夹（格式为`user_name: friend1 friend2 ...`）。任务目标是为每个用户推荐最多5个未关注的好友候选，利用两阶段MapReduce作业完成。总体流程如下：

- 第一阶段 (Job1): 从互相关注用户对生成“朋友的朋友”候选对。
  - Map阶段设计: 输入为互相关注用户对`userA-userB`, 输出为互相关注关系生成双向键值对，即`(userA, userB)`和`(userB, userA)`，表示双向好友关系。
  - Reduce阶段设计: 收集用户(例如Alice)的所有直接好友, 对好友列表两两配对, 例如收集`(Alice, Bob)`和`(Alice, Changxue)`为`(Alice, [Bob, Changxue])`, 输出`(Bob, Changxue), (Changxue, Bob)`
- 第二阶段 (Job2): 结合关注列表过滤已关注用户，保留每个用户最多5个推荐候选。
  - Map阶段设计: 输入为**Job1的输出**(候选对)以及**关注列表文件**, 分别解析候选对和关注列表, 进行输出, 例如:
    - 输入候选对为`(Bob, Changxue)`输出为`(Bob, C:Changxue)`
    - 输入关注列表为`Bob: Alice David`; 输出为`(Bob, F:Alice,David)`
  - Reduce阶段设计: 输入为分组后的键值对，键为用户，值为候选列表（C:）和关注列表（F:）
    - 分离已关注用户（F:）和候选用户（C:）, 从候选用户中移除已关注用户
    - 保留最多5个候选用户，格式化为逗号分隔的字符串
- 输出: 生成推荐结果，格式为`user: candidate1,candidate2,...`

核心代码

Job1

Mapper阶段的核心代码:

```
String line = value.toString().trim();
if (line.isEmpty() || !line.contains("-")) return;
String[] parts = line.split("-", 2);
if (parts.length != 2) return;
// A-B 互相关注，双向输出
```

```
String a = parts[0].trim();
String b = parts[1].trim();
if (a.isEmpty() || b.isEmpty()) return;

user.set(a); friend.set(b);
ctx.write(user, friend);

user.set(b); friend.set(a);
ctx.write(user, friend);
```

Reducer阶段的核心代码:

```
// 收集所有直接好友
List<String> friends = new ArrayList<>();
for (Text t : vals) {
    friends.add(t.toString());
}
// 两两配对，生成候选推荐对：如果 A 与 B 和 C 互粉，则输出 B->C 和 C->B
int n = friends.size();
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (i == j) continue;
        String b = friends.get(i);
        String c = friends.get(j);
        outKey.set(b);
        outVal.set(c);
        ctx.write(outKey, outVal);
    }
}
```

Job2

Mapper阶段的核心代码:

```
/*** === Job2 MapperA: 读取 Job1 的候选对 (userB \t userC) === */
String line = value.toString().trim();
if (line.isEmpty()) return;
String[] parts = line.split("\\t");
if (parts.length != 2) return;
String b = parts[0].trim();
String c = parts[1].trim();
if (b.isEmpty() || c.isEmpty()) return;

user.set(b);
tagAndVal.set("C:" + c);
ctx.write(user, tagAndVal);

/*** === Job2 MapperB: 读取关注列表 (user: f1 f2 ...) === */
String line = value.toString().trim();
if (!line.contains(":")) return;
String[] parts = line.split(":", 2);
```

```
String u = parts[0].trim();
String[] fs = parts[1].trim().split("\\s+");
if (u.isEmpty()) return;

// 将整个列表当成一个值
user.set(u);
tagAndVal.set("F:" + String.join(",", fs));
ctx.write(user, tagAndVal);
```

Reducer阶段的核心代码:

```
// 收集已关注列表
Set<String> followed = new HashSet<>();
// 收集所有候选
Set<String> candidates = new LinkedHashSet<>();
// 解析输入
for (Text t : vals) {
    String s = t.toString();
    if (s.startsWith("F:")) {
        String[] fs = s.substring(2).split(",");
        for (String f : fs) if (!f.isEmpty())
            followed.add(f);
    } else if (s.startsWith("C:")) {
        String c = s.substring(2);
        if (!c.isEmpty()) {
            candidates.add(c);
        }
    }
}
// 过滤掉已关注的
candidates.removeAll(followed);
// 取前5
Iterator<String> it = candidates.iterator();
List<String> out = new ArrayList<>();
int cnt = 0;
while (it.hasNext() && cnt < 5) {
    out.add(it.next());
    cnt++;
}

if (!out.isEmpty()) {
    result.set(String.join(",", out));
    ctx.write(user, result);
}
```

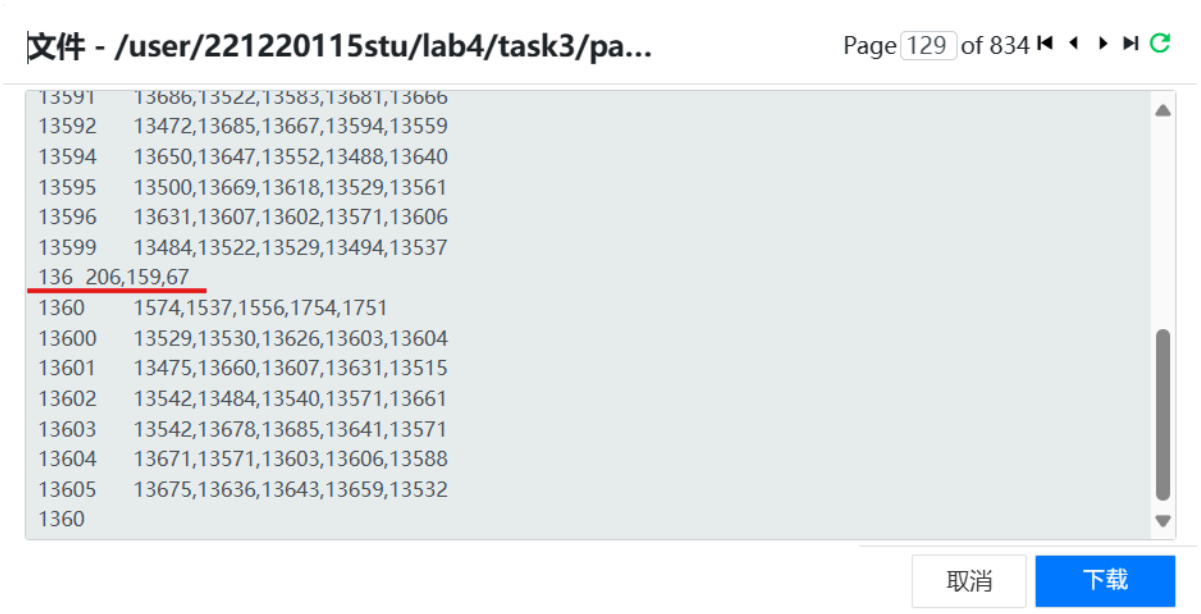
执行过程及结果

在集群上输入以下命令执行任务:

```
yarn jar /home/221220115stu/task1-1.0-SNAPSHOT.jar \
com.example.FriendRecommendation \
/user/221220115stu/lab4/task1/part-r-00000 \
/user/root/Exp4 \
/user/221220115stu/lab4/tmp \
/user/221220115stu/lab4/task3
```

输出结果文件位于`/user/221220115stu/lab4/task3`

输出结果的部分截图如下:



执行报告