

# 《分布式编程模型与系统》期末考查作业

学号	姓名	成绩
10195501440	徐涵钰	

## 实验目的

考察影响MapReduce性能的因素

## 设计思路

1. MapReduce需要对map结果进行shuffle，shuffle过程会进行大量的磁盘读写，加入Combine先对键相同的键值对做归并，再写入到磁盘，可以减少磁盘IO
2. MapReduce的shuffle阶段的磁盘读写，可以考虑通过将map结果进行压缩，再存储到磁盘，从而减少map写和reduce读的数据量
3. 考虑从Java编程角度出发，重用变量，以减少对节点资源的消耗

## 实验设置

名称	设置
本机操作系统	Windows 10
IDEA	Intellij IDEA 2022.1
JDK	JDK 1.8
云端总节点数	2
Hadoop版本	2.10.1
数据集1	pd.train
数据集1大小	2.02GB
数据集2	pd.valid
数据集2大小	8.25MB

## 实验过程

### combine的影响

针对思路1进行验证，编写了带有combine和不带combine的WordCount应用，提交到分布式部署的云端，在<http://ecnu01:19888>端口查看运行时间的差异。

1. 编写java代码

带有Combine和不带Combine的java代码区别：

不带Combine，则在WordCount中只设置map和reduce方法：

```
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);
```

带Combine，则在WordCount中要对combine方法也进行设置：

```
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);
job.setCombinerClass(WordCountCombiner.class);
```

并且写一个名为WordCountCombiner的方法，继承Reducer方法，实现对每个具有相同键值对的值进行计数，输出类型为Text和IntWritable键值对：

```
int sum=0; //初始一个计数器
for(IntWritable value:values){
    sum += value.get(); //对values进行遍历，每次加1
}
context.write(key, new IntWritable(sum));
```

在本地运行之后，将两个应用分别打包为jar包，命名为wordcount.jar和CWordCount.jar，分别对应不含combine以及含有combine的应用

## 2. 云端部署

准备了两台机器，配置其中一台ecnu01为主节点+从节点，另一台ecnu02为从节点+客户端。HDFS的input文件夹中存储了pd.train数据集。通过Xftp直接将两个jar包都上传到hadoop-2.10.1/myApp目录下，随后在主节点上启动Yarn服务和HDFS服务，在客户端提交jar包，运行应用

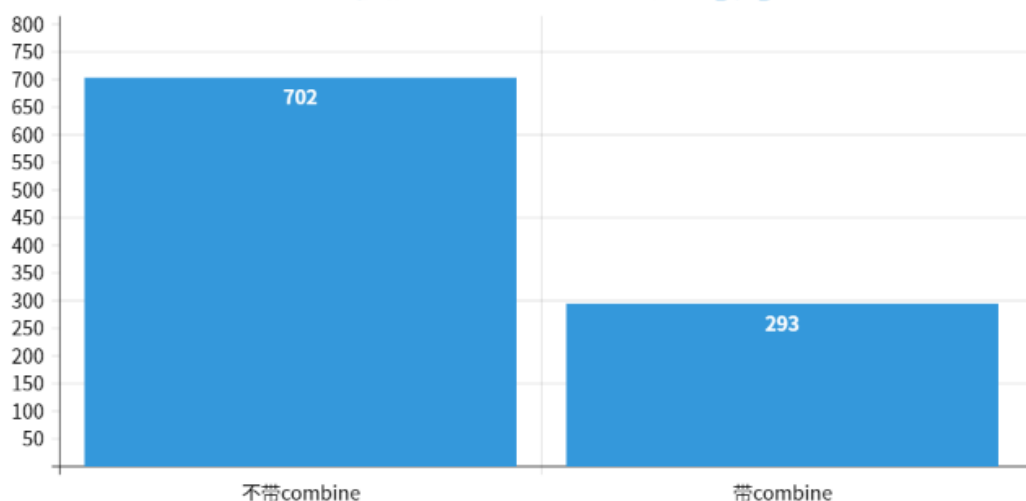
## 3. 性能对比

访问<http://ecnu01:19888>，能够看到历史记录如下：

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reducers Total	Reducers Completed	Elapsed Time
2022.06.23 20:05:45 CST	2022.06.23 20:05:51 CST	2022.06.23 20:17:33 CST	job_1655974675552_0010	WordCount	ubuntu	default	SUCCEEDED	17	17	1	1	00hrs, 11mins, 42sec
2022.06.23 17:44:42 CST	2022.06.23 17:44:48 CST	2022.06.23 17:49:41 CST	job_1655974675552_0009	WordCount	ubuntu	default	SUCCEEDED	17	17	1	1	00hrs, 04mins, 53sec

其中，job\_1655974675552\_0010对应未使用combine的应用，job\_1655974675552\_0009对应使用了combine的应用。可以看到，未使用combine的应用用时11min42s，而使用了combine的应用用时04min53s，为未使用combine情况下用时的41%。

## 是否使用combine的用时对比



## 压缩的影响

针对思路2进行验证，修改WordCount代码，使其对map的输出做压缩，在本地运行后，比较shuffle数据量的差异。

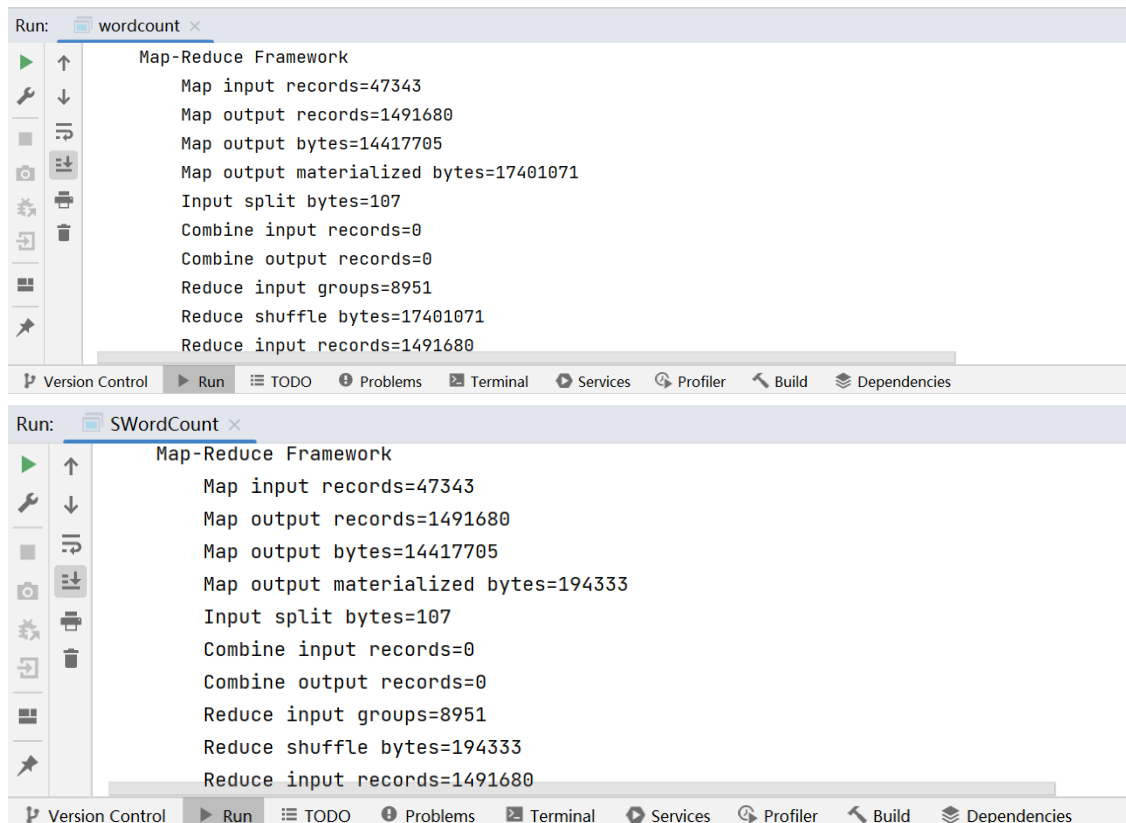
### 1. 修改代码

修改WordCount.java，设置对map的输出做压缩。在这里为了便于本地运行，选择了hadoop自带的bzip2压缩格式。

```
// 设置对map方法输出做压缩
Configuration configuration = new Configuration();
// 开启map端输出压缩
configuration.setBoolean("mapreduce.map.output.compress", true);
// 设置map端输出压缩方式
configuration.set("mapreduce.map.output.compress.codec", "org.apache.hadoop.io.compress.BZip2Codec");
Job job = Job.getInstance(configuration, getClass().getSimpleName());
```

### 2. shuffle量比较

分别运行原本的wordcount和加入压缩过后的wordcount，使用pd.valid作为输入，分别得到运行信息如下：



The image displays two screenshots of the IntelliJ IDEA Run console, comparing the performance of a standard WordCount job versus a compressed SWordCount job. Both jobs use 'pd.valid' as input.

**Run: wordcount**

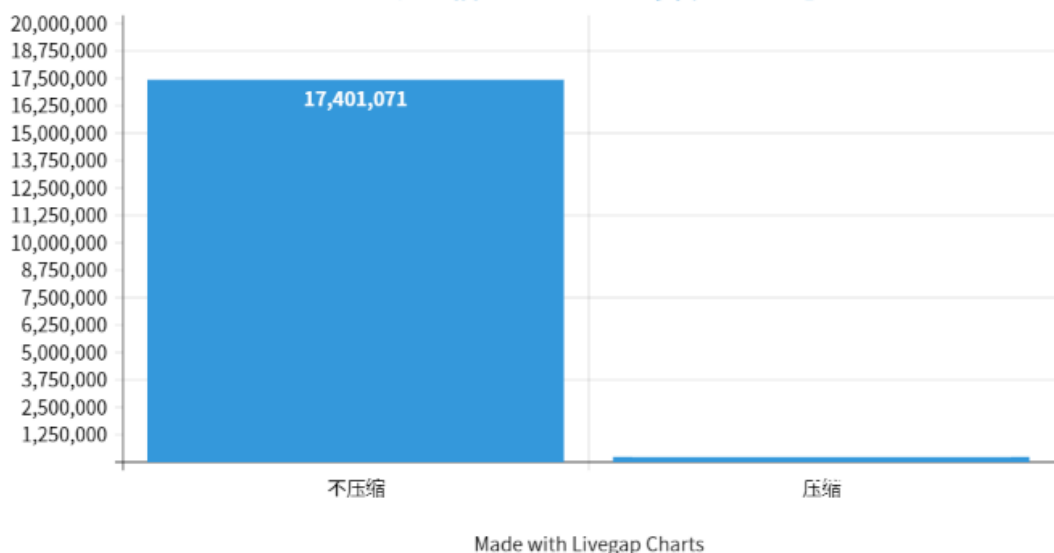
- Map input records=47343
- Map output records=1491680
- Map output bytes=14417705
- Map output materialized bytes=17401071
- Input split bytes=107
- Combine input records=0
- Combine output records=0
- Reduce input groups=8951
- Reduce shuffle bytes=17401071
- Reduce input records=1491680

**Run: SWordCount**

- Map input records=47343
- Map output records=1491680
- Map output bytes=14417705
- Map output materialized bytes=194333
- Input split bytes=107
- Combine input records=0
- Combine output records=0
- Reduce input groups=8951
- Reduce shuffle bytes=194333
- Reduce input records=1491680

可以看到，不使用压缩的wordcount中，reduce shuffle bytes为17401071，而使用了压缩的SWordCount中，reduce shuffle bytes为194333，为压缩前所需shuffle量的1.1%，大大减少了磁盘IO以及shuffle过程中的网络IO。

## 是否使用压缩的shuffle数据量对比

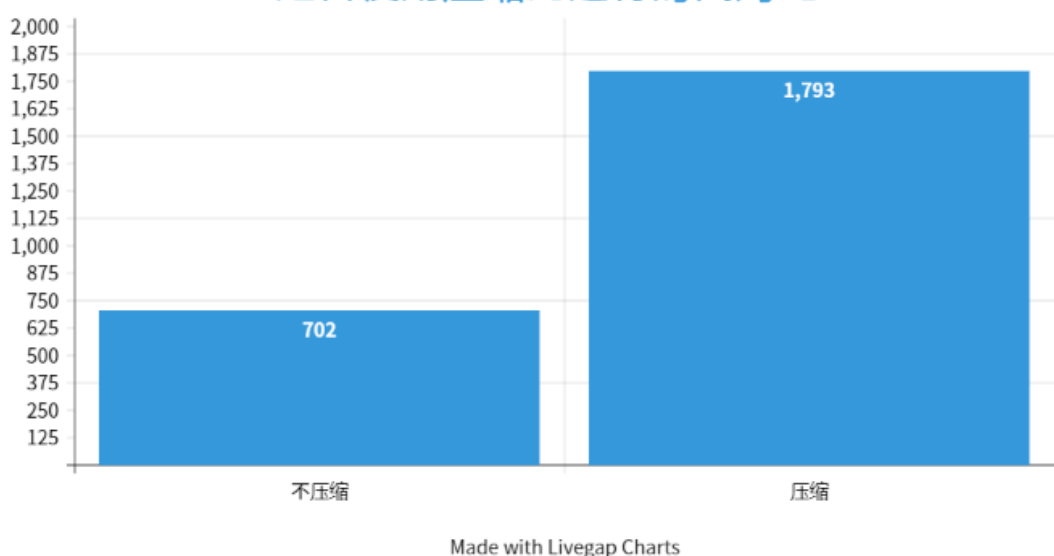


### 3. 运行时长比较

同样，将该程序打包后，上传到客户端，并在客户端通过提交jar包的方式，运行该程序。结果发现，该程序运行时长相比起原本版本大大增加：

Started:	Fri Jun 24 20:40:09 +0800 2022
Launched:	Fri Jun 24 20:40:10 +0800 2022
Finished:	Fri Jun 24 21:10:03 +0800 2022
Elapsed:	29mins, 53sec

## 是否使用压缩的运行时间对比



## 变量的影响

对思路3进行验证，修改代码，同样提交到分布式部署的云端，在<http://ecnu01:19888>端口查看其与修改前的wordcount运行时间的差异。

### 1. 修改代码

以map方法为例，原本的wordcount采用的方法如下，这种方法每次都新建一个Text类型和IntWritable类型的变量，造成对堆空间的浪费：

```
context.write(new Text(data), new IntWritable(1));
```

修改代码如下：

```

Text text = new Text();
IntWritable intWritable = new IntWritable(1);

@Override
protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    String[] datas = value.toString().split(" ");
    for (String data : datas) {
        text.set(data);
        context.write(text, intWritable);
    }
}

```

首先建立了两个名为text和intWritable的全局变量，随后的map方法仅对其进行修改，而不会新建变量

## 2. 云端运行

云端的部署与先前一致，同样通过xftp将jar包上传至客户端，随后客户端通过提交jar包运行程序

## 3. 运行结果

访问<http://ecnu01:8088>端口，查看作业具体信息，可以看到该作业运行时间如下，为11min6s，相比之前的11min42s没有显著提升：

FinalStatus Reported by AM:	SUCCEEDED
Started:	Fri Jun 24 17:46:18 +0800 2022
Launched:	Fri Jun 24 17:46:19 +0800 2022
Finished:	Fri Jun 24 17:57:25 +0800 2022
Elapsed:	11mins, 6sec
Tracking URL:	<a href="#">History</a>
Log Aggregation Status:	SUCCEEDED

考虑原因可能为各个task分配的堆大小不够小，难以体现差距。但是我将mapreduce.map.java.opts、mapreduce.reduce.java.opts两个参数设置为512M后，不断出现Container killed，程序无法正常运行。

# 结论

1. 在有无combine对mapreduce执行时间的影响中，观察到了二者间显著的差别，符合预期。原因分析如下：

combine在map任务和reduce任务之间执行，相当于对一个缓冲区中的map输出做了局部reduce。举例分析可知，如果原本的某一节点的map任务输出了3个[A, 1]，则若没有combine，这三个输出都将通过shuffle，传送到reduce任务；但是若有combine，这3个[A, 1]将合并为1个[A, 3]，再通过shuffle传输给reduce任务。于是，原本需要传输3个的变为传输1个。由此类推，使用combine能够将同一缓冲区的所有相同键的键值对做合并，从而减少了shuffle过程中传输的数据量，减少时间开销。

2. 在通过压缩减少shuffle数据量从而提高性能的实验中，shuffle数据量确有减少，但是运行时间大幅增加，不符合预期。原因分析如下：

在压缩算法中选用了Gzip2，该压缩方法虽然为hadoop本身支持，且压缩率高，但是有压缩/解压速度慢的缺点。在本实验中，将其用作map输出和reduce输入，导致其急需要压缩也需要解压，耗费时间长。

3. 在java变量对性能的影响中，运行时间没有显著减少，与预期不符。原因分析如下：

默认配置为有1GB大小的堆大小，所以垃圾回收机制没有成功启动。认为若将堆大小设置得更小时能够观察到变化，但目前mapreduce作业总是崩溃，未能成功实验验证。

github链接：[Audrey1349/distributed \(github.com\)](https://github.com/Audrey1349/distributed)

