

华东师范大学数据学院上机实践报告

| | | |
|------------------------|----------------|------------------|
| 课程名称：分布式模型与编程 | 年级：2018 | 上机实践成绩： |
| 指导教师：徐辰 | | 姓名：孙秋实 |
| 上机实践名称：MapReduce2.x 编程 | 学号：10185501402 | 上机实践日期：2021/3/29 |
| 上机实践编号：La6 | 组号：Group5 | 上机实践时间： |

Part 1

实验目的

- (1) 学习编写简单的基于 Java API 的 Hadoop2.x Map Reduce 程序
- (2) 分别在单机集中式、单机伪分布式、分布式模式下运行 MapReduce 相关的程序

Part 2

实验任务

- (1) 完成基于 Java API 编写的 WordCount 的 MapReduce 程序；
- (2) 在单机集中式、单机伪分布式、分布式模式下调试运行改程

Part 3

使用环境

- (1) 操作系统：Ubuntu 18.04
- (2) JDK 版本：1.8
- (3) Hadoop 版本：2.10.1
- (4) IDEA 2020.2.3

Part 4

实验过程

Section 1

编写 MapReduce 应用程序

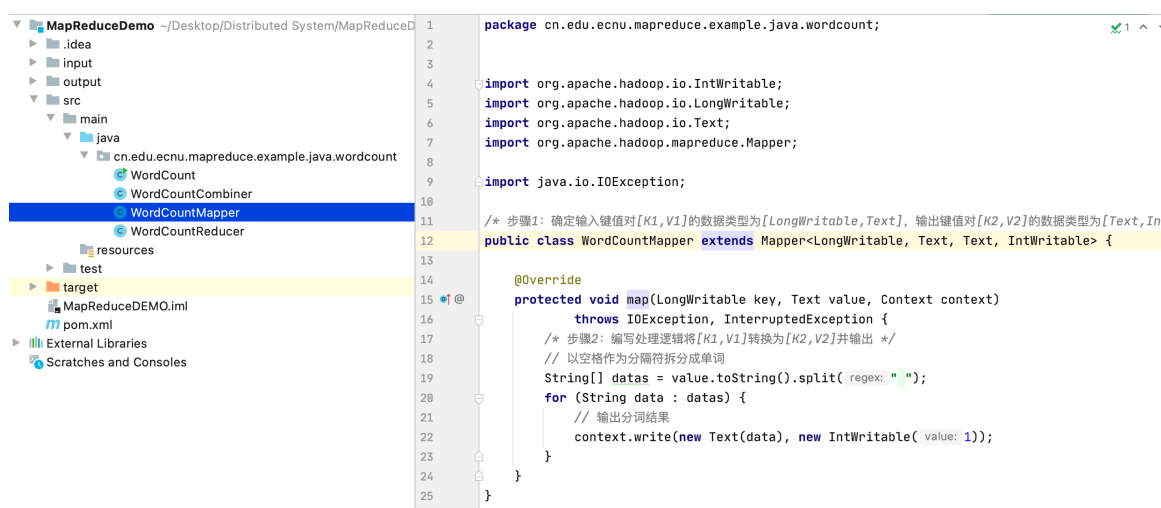
首先是建立一个 Maven 项目并且添加依赖

- 新建 MapReduceDemo 的 maven 项目
- 编辑 pom.xml，添加相关的依赖包

```
<dependencies>
|   <dependency>
|       <groupId>org.apache.hadoop</groupId>
|       <artifactId>hadoop-common</artifactId>
|       <version>2.10.1</version>
|   </dependency>
|
|   <dependency>
|       <groupId>org.apache.hadoop</groupId>
|       <artifactId>hadoop-hdfs</artifactId>
|       <version>2.10.1</version>
|   </dependency>
|
|   <dependency>
|       <groupId>org.apache.hadoop</groupId>
|       <artifactId>hadoop-client</artifactId>
|       <version>2.10.1</version>
|   </dependency>
| </dependencies>
```

图 1: 为新建 maven 项目添加依赖

以下为 WordCount 程序的三个组件，代码参照了实验手册中提供的 GitHub 仓库的代码



```
package cn.edu.ecnu.mapreduce.example.java.wordcount;

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.IOException;

/* 步骤1: 确定输入键值对 [K1, V1] 的数据类型为 [LongWritable, Text], 输出键值对 [K2, V2] 的数据类型为 [Text, IntWritable] */
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        /* 步骤2: 编写处理逻辑将 [K1, V1] 转换为 [K2, V2] 并输出 */
        // 以空格作为分隔符拆分成单词
        String[] datas = value.toString().split(" ");
        for (String data : datas) {
            // 输出分词结果
            context.write(new Text(data), new IntWritable(1));
        }
    }
}
```

图 2: WordCount Mapper 程序

华东师范大学数据科学与工程学院学生实验报告

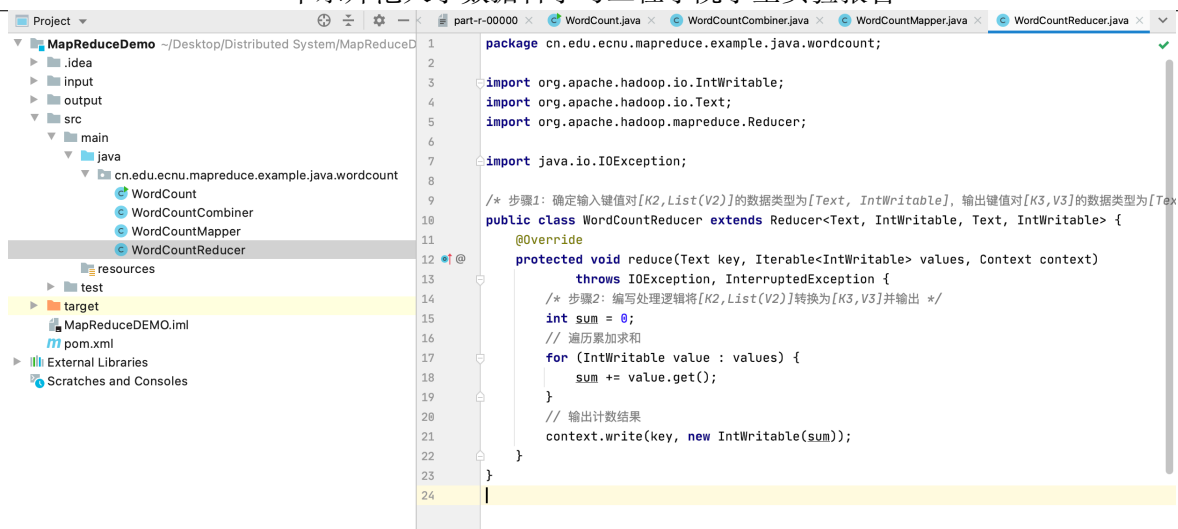


图 3: WordCount Reducer 程序

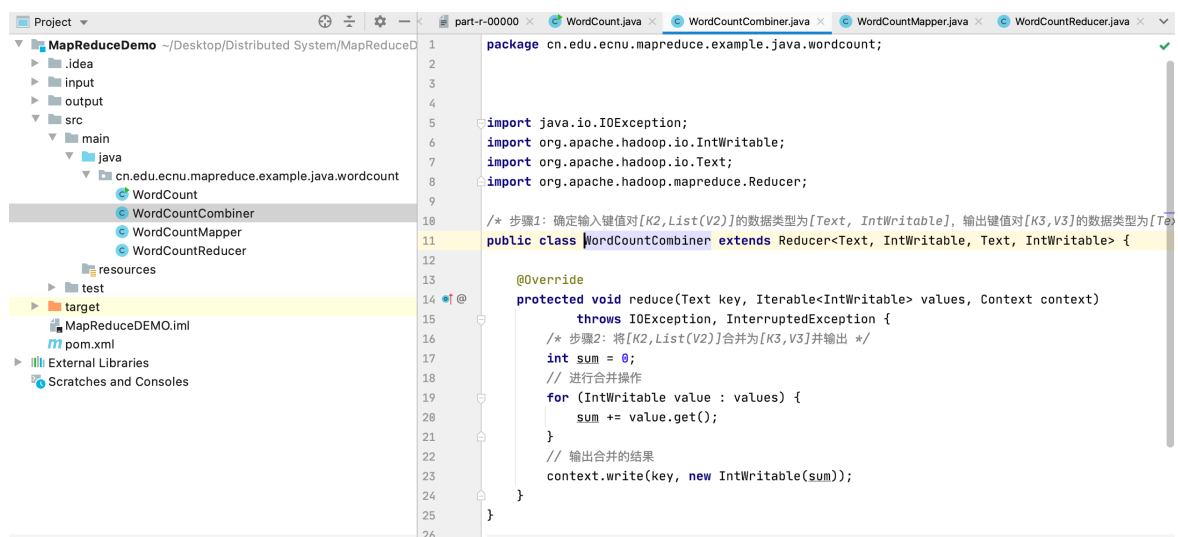
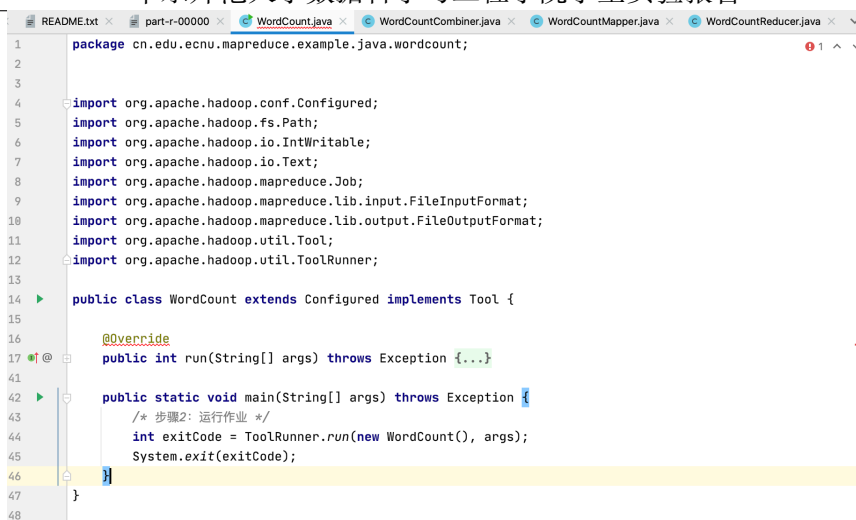


图 4: WordCount Combiner 程序

最后是 WordCount 程序的主方法



```
1 package cn.edu.ecnu.mapreduce.example.java.wordcount;
2
3
4 import org.apache.hadoop.conf.Configured;
5 import org.apache.hadoop.fs.Path;
6 import org.apache.hadoop.io.IntWritable;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
9 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11 import org.apache.hadoop.util.Tool;
12 import org.apache.hadoop.util.ToolRunner;
13
14 public class WordCount extends Configured implements Tool {
15
16     @Override
17     public int run(String[] args) throws Exception {...}
18
19     public static void main(String[] args) throws Exception {
20         /* 步骤2: 运行作业 */
21         int exitCode = ToolRunner.run(new WordCount(), args);
22         System.exit(exitCode);
23     }
24 }
```

图 5: WordCount 主方法

Section 2

调试 MapReduce 程序（本地）

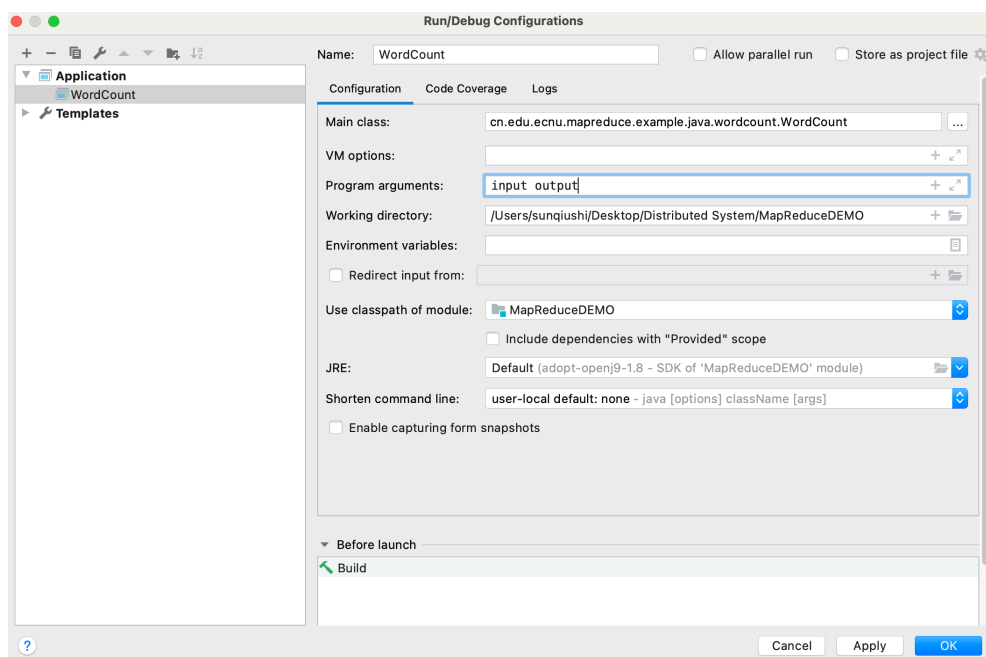


图 6: 运行参数配置界面

Section 3

运行 MapReduce 程序（本地）

（使用了助教提供的实验六补充文档中的数据示例）

```

pom.xml (MapReduceDEMO) x README.txt x part-r-00000 x WordCountCombiner.java x WordCountMapper.java x
1 apple 1
2 applepen 1
3 boy 2
4 cat 3
5 dog 4
6 pen 1
7 pineapple 1
8 pineapplepen 1
9 |
    
```

图 7: WordCount 的运行结果

Section 4

运行 MapReduce 程序（单机伪分布式）

与上一个任务一样，把 IDEA 中的项目打包为 jar 文件，通过 scp 命令提交到虚拟机的文件系统中，随后启动 Yarn 和 HDFS 任务，进行词频统计，结果如下所示

```

18
(BIS), 1
(ECCN) 1
(TSU) 1
(see 1
5D002.C.1, 1
740.13) 1
<http://www.wassenaar.org/> 1
Administration 1
Apache 1
BEFORE 1
BIS 1
Bureau 1
    
```

图 8: 伪分布式环境下提交任务

这个程序实现了统计 HDFS 中 dase-local/input 路径下的 README.txt 文件中的单词数量的任务，接下来我们在分布式模式下执行它，看看效果如何。（但是一定要记得，把本地的 Yarn 服务和 HDFS 服务关闭再开始分布式实验！）

Section 5

运行 MapReduce 程序（分布式）

接下来，在分布式环境中执行词频统计任务，首先是授权，再客户端把打好包的项目传输到 dase-dis/hadoop-2.10.1/myApp 目录下

```

dase-dis@ecnu04:~/.ssh$ su dase-dis
Password:
dase-dis@ecnu04:~/.ssh$ mkdir ~/hadoop-2.10.1/myApp/
    
```

图 9: 授权 dase-dis

```

tangqiong@tangqiongdeMacBook-Pro Desktop % scp WordCount.jar dase-dis@10.24.21.106:/home/dase-dis/hadoop-2.10.1/
myApp/
dase-dis@10.24.21.106's password:
WordCount.jar
tangqiong@tangqiongdeMacBook-Pro Desktop % 100% 4589 522.6KB/s 00:00
    
```

图 10: 客户端提交任务

在任务执行时，可以在从节点看到自己担任的角色正在运行哪些程序协助这个分布式计算过程

```
dase-dls@ecnu03:~$ jps
9910 NodeManager
10073 Jps
9098 DataNode
dase-dls@ecnu03:~$
```

图 11: 任务运行时的从节点状态

```
dase-dls@ecnu04:~/hadoop-2.10.15$ ./bin/hadoop jar ./myApp/WordCount.jar input/README.txt output
21/03/29 21:30:26 INFO client.RMProxy: Connecting to ResourceManager at ecnu01/10.24.21.116:8032
21/03/29 21:30:27 INFO input.FileInputFormat: Total input files to process : 1
21/03/29 21:30:27 INFO mapreduce.JobSubmitter: number of splits:1
21/03/29 21:30:27 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1617024132543_0001
21/03/29 21:30:27 INFO conf.Configuration: resource-types.xml not found
21/03/29 21:30:27 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
21/03/29 21:30:27 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = MB, type
= COUNTABLE
21/03/29 21:30:27 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = CO
UNTABLE
21/03/29 21:30:27 INFO impl.YarnClientImpl: Submitted application application_1617024132543_0001
21/03/29 21:30:27 INFO mapreduce.Job: The url to track the job: http://ecnu01:8088/proxy/application_161
7024132543_0001/
21/03/29 21:30:27 INFO mapreduce.Job: Running job: job_1617024132543_0001
21/03/29 21:30:33 INFO mapreduce.Job: Job job_1617024132543_0001 running in uber mode : true
21/03/29 21:30:33 INFO mapreduce.Job: map 100% reduce 100%
21/03/29 21:30:34 INFO mapreduce.Job: Job job_1617024132543_0001 completed successfully
21/03/29 21:30:34 INFO mapreduce.Job: Counters: 52
File System Counters
  FILE: Number of bytes read=3718
  FILE: Number of bytes written=5593
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=3018
  HDFS: Number of bytes written=440692
  HDFS: Number of read operations=35
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=8
```

图 12: 进行词频统计

最后可以查看词频统计的结果，发现和伪分布式模式下得到的结果是相同的。

```
dase-dls@ecnu04:~/hadoop-2.10.15$ ./bin/hdfs dfs -cat output/p*
18
(BIS), 1
(ECCN) 1
(TSU) 1
(see 1
SD002.C.1, 1
740.13) 1
<http://www.wassenaar.org/> 1
Administration 1
Apache 1
BEFORE 1
BIS 1
Bureau 1
Commerce, 1
Commodity 1
Control 1
Core 1
Department 1
ENC 1
Exception 1
Export 2
For 1
```

图 13: 分布式模式下词频统计结果

分布式下执行的速度非常快，效率远超过伪分布式和集中式的部署方法。

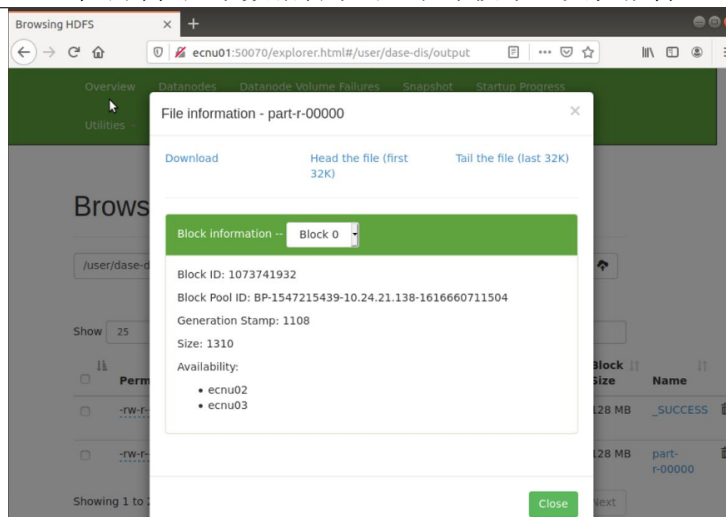


图 14: 分布式模式下运行结果存储位置

Part 5

思考题

Section 1

如何从 Hadoop Job History 的 Web UI 中查看某个 Map Reduce 应用程序分别启动了多少个 Map 任务和 Reduce 任务?

Answer: 如下图所示，一个 Map 任务，一个 Reduce 任务

| Job ID | Name | User | Queue | State | Maps Total | Maps Completed | Reduces Total | Reduces Completed | Elapsed Time |
|------------------------|-------------|----------|---------|-----------|------------|----------------|---------------|-------------------|---------------------|
| job_1617024132543_0001 | WordCount | dase-dis | default | SUCCEEDED | 1 | 1 | 1 | 1 | 00hrs, 00mins 01sec |
| job_1616671487701_0003 | word count | dase-dis | default | SUCCEEDED | 17 | 17 | 1 | 1 | 00hrs, 02mins 04sec |
| job_1616671487701_0002 | grep-sort | dase-dis | default | SUCCEEDED | 1 | 1 | 1 | 1 | 00hrs, 00mins 01sec |
| job_1616671487701_0001 | grep-search | dase-dis | default | SUCCEEDED | 30 | 30 | 1 | 1 | 00hrs, 00mins 26sec |

图 15: 思考题 1-1

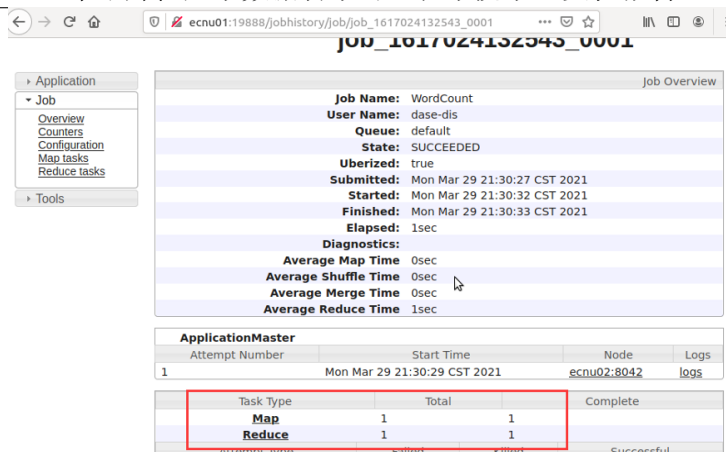


图 16: 思考题 1-2

Section 2

如何从 Hadoop Job History 的 Web UI 中查看某个 Map Reduce 应用程序启动的 Map 任务和 Reduce 任务分别是在哪些 NodeManager 上执行的?

Answer: Hadoop Job History 的 Web UI 可以查看相应的 Node 编号

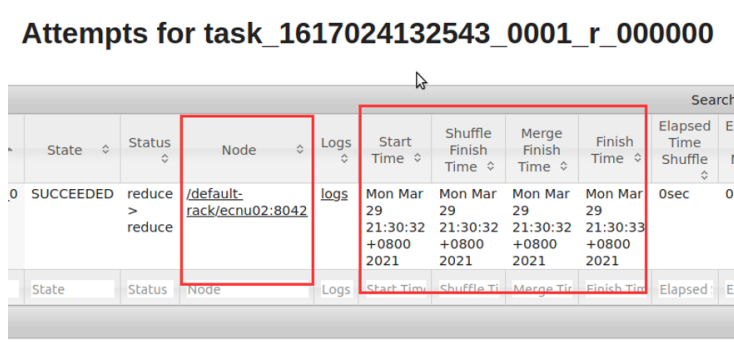


图 17: 思考题 2

Section 3

对于一个 Map Reduce 应用程序, 是否存在某个时刻 Map 任务和 Reduce 任务同时在运行? 结合 Hadoop Job History 的 Web UI 给出例证。

存在某个时刻 Map 任务和 Reduce 任务同时在运行, 如下图所示, 图 3-2 的 Reduce Task 发生时 Map Task 还没有全部完成 (虽然在这个任务上二者重合的时间很短)

| Task ID | State | Start Time | Finish Time |
|----------------------------------|-----------|--------------------------------|--------------------------------|
| task_1616671487701_0003_m_000002 | SUCCEEDED | Thu Mar 25 21:16:57 +0800 2021 | Thu Mar 25 21:18:13 +0800 2021 |
| task_1616671487701_0003_m_000003 | SUCCEEDED | Thu Mar 25 21:16:57 +0800 2021 | Thu Mar 25 21:18:12 +0800 2021 |
| task_1616671487701_0003_m_000004 | SUCCEEDED | Thu Mar 25 21:16:57 +0800 2021 | Thu Mar 25 21:18:09 +0800 2021 |
| task_1616671487701_0003_m_000005 | SUCCEEDED | Thu Mar 25 21:16:56 +0800 2021 | Thu Mar 25 21:18:10 +0800 2021 |
| task_1616671487701_0003_m_000006 | SUCCEEDED | Thu Mar 25 21:16:56 +0800 2021 | Thu Mar 25 21:18:24 +0800 2021 |
| task_1616671487701_0003_m_000007 | SUCCEEDED | Thu Mar 25 21:16:57 +0800 2021 | Thu Mar 25 21:18:27 +0800 2021 |
| task_1616671487701_0003_m_000008 | SUCCEEDED | Thu Mar 25 21:16:57 +0800 2021 | Thu Mar 25 21:18:29 +0800 2021 |
| task_1616671487701_0003_m_000009 | SUCCEEDED | Thu Mar 25 21:16:56 +0800 2021 | Thu Mar 25 21:18:27 +0800 2021 |
| task_1616671487701_0003_m_000010 | SUCCEEDED | Thu Mar 25 21:16:56 +0800 2021 | Thu Mar 25 21:18:27 +0800 2021 |

图 18: 思考题 3-1

| Task ID | State | Start Time | Finish Time |
|----------------------------------|-----------|--------------------------------|--------------------------------|
| task_1616671487701_0003_r_000000 | SUCCEEDED | Thu Mar 25 21:18:13 +0800 2021 | Thu Mar 25 21:18:58 +0800 2021 |

图 19: 思考题 3-2

Part 6

实验总结

思考题中涉及了一些 mapreduce 在处理如词频统计任务时的特性，我们做一些进一步的研究

MapTask 的并行度

一个 job 的 mapTask 并行度由客户端在提交 job 时决定。

将待处理数据执行逻辑切片，即按照一个特定切片大小，将待处理数据划分成逻辑上的多个 split，然后每一个 split 分配一个 mapTask 并行实例处理。

注意! block 是 HDFS 上物理上存储的存储的数据，切片是对数据逻辑上的划分。两者之间没有关系。即使 HDFS 上是 128M 存储，Mapreduce 也会切片，只是默认切片也是 128M。也可以非 128M 切片，如 100M，多余的部分由框架内部处理和其他结点进行拼接切片。因为切片数量决定了 mapTask 进程数量。

一些实验中可能会遇到的重要的参数：

block Size: (这个其实在上一次实验中接触过)Hadoop 2.x 默认的 block 大小是 128MB，Hadoop 1.x 默认的 block 大小是 64MB，可以在 hdfs-site.xml 中设置 dfs.block.size，注意单位是 byte。

split 分片大小: 范围在 mapred-site.xml 中设置，mapred.min.split.size mapred.max.split.size，minSplitSize 大小默认为 1B，maxSplitSize 大小默认为 Long.MAX_VALUE (一个极大值)

MapTask 数量: Hadoop 提供了一个设置 map 进程个数的参数 `mapred.map.tasks`, 我们可以通过这个参数来控制 map 的个数。但是通过这种方式设置 map 的进程个数, 并不是每次都有效的。原因是 `mapred.map.tasks` 只是一个 hadoop 的参考数值, 最终 map 的进程个数, 还取决于很多因素。需要注意, Mapreduce 的每一个 map 进程处理的数据是不能跨越文件的。