1. 需求概述

示例需求主要是基于全球的气象数据进行数据挖掘，求出每年的最高的气温。

2. 数据准备

有了需求了那么必须得有数据，需要下载气象数据。

2.1. 数据下载

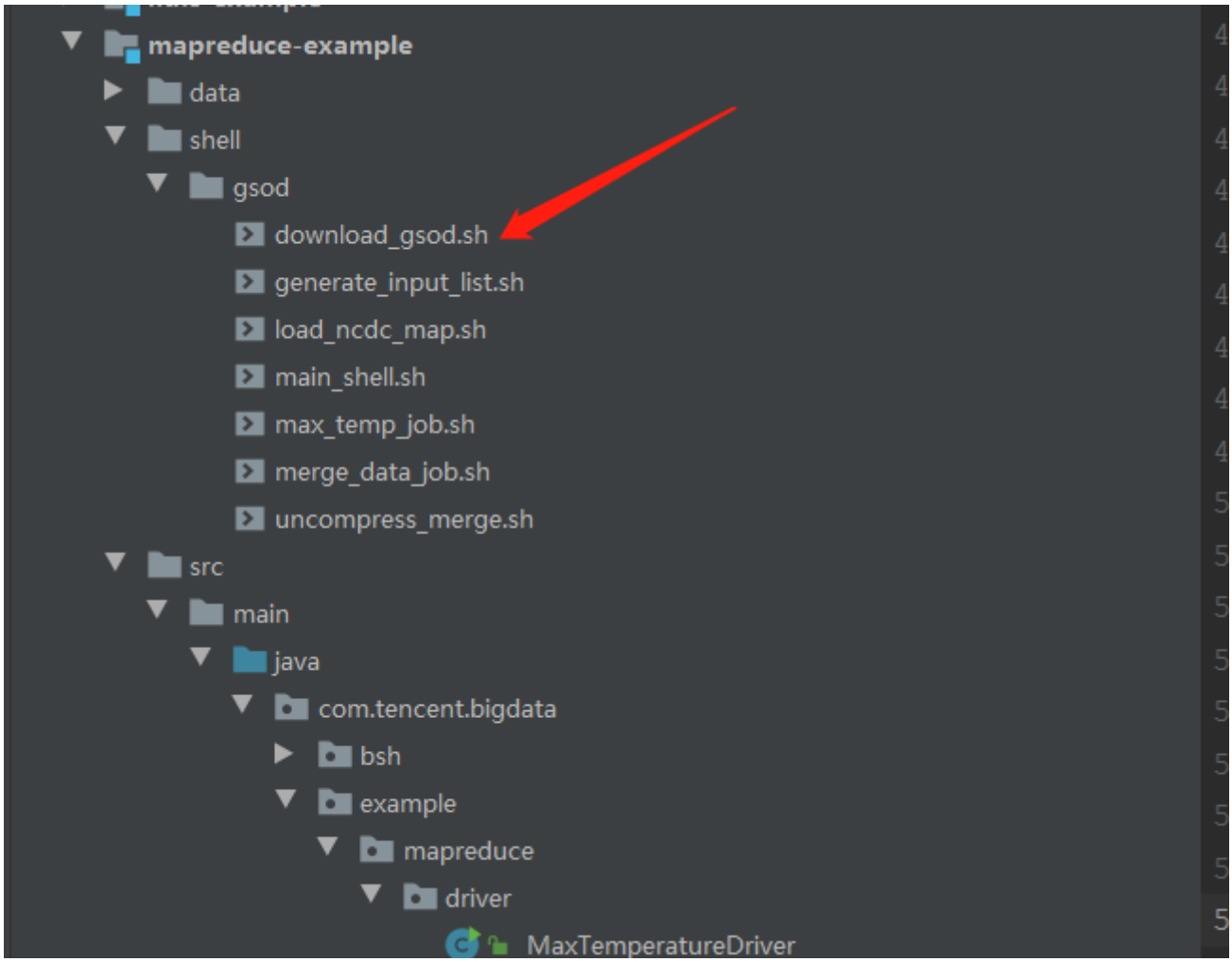
气象数据是通过全球各个气象站收集并上传到数据中心汇集的，气象数据是公开的数据。

可以直接NCDC点击下载地址进行下载。

这里是测试只加载几年的数据即可，下载1949-1963年数据到本地/home/hadoop/gsod/data目录下。

2.2. 下载命令

下载命令在工程的shell/gsod/download_gsod.sh下，如下图



下载命令如下图：

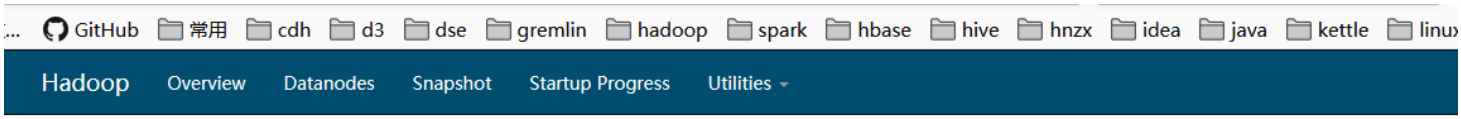
```
#!/bin/bash
for i in {1949..1963}
do
cd /home/hadoop/gsod/data || exit
wget --execute robots=off -r -np -nH --cut-dirs=4 --accept=gsod* -R index.html* ftp://ftp.ncdc.noaa.gov/pub/data/gsod/$i/
done
```

3. 数据上传

> 本地数据上传到HDFS文件系统，命令如下：

```
# ncdc原始数据文件上传到hdfs
hdfs dfs -put /home/hadoop/gsod/data/* /data/gsod
```

> HDFS上查看数据情况



Browse Directory

/data/gsod

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	10.56 MB	Sun Sep 22 21:46:21 +0800 2019	3	128 MB	gsod_1949.tar
-rw-r--r--	root	supergroup	11.82 MB	Sun Sep 22 21:46:21 +0800 2019	3	128 MB	gsod_1950.tar
-rw-r--r--	root	supergroup	12.04 MB	Sun Sep 22 21:46:21 +0800 2019	3	128 MB	gsod_1951.tar
-rw-r--r--	root	supergroup	13.34 MB	Sun Sep 22 21:46:21 +0800 2019	3	128 MB	gsod_1952.tar
-rw-r--r--	root	supergroup	14.59 MB	Sun Sep 22 21:46:21 +0800 2019	3	128 MB	gsod_1953.tar
-rw-r--r--	root	supergroup	15.41 MB	Sun Sep 22 21:46:21 +0800 2019	3	128 MB	gsod_1954.tar
-rw-r--r--	root	supergroup	15.2 MB	Sun Sep 22 21:46:22 +0800 2019	3	128 MB	gsod_1955.tar
-rw-r--r--	root	supergroup	15.74 MB	Sun Sep 22 21:46:22 +0800 2019	3	128 MB	gsod_1956.tar
-rw-r--r--	root	supergroup	19.85 MB	Sun Sep 22 21:46:22 +0800 2019	3	128 MB	gsod_1957.tar
-rw-r--r--	root	supergroup	21.83 MB	Sun Sep 22 21:46:22 +0800 2019	3	128 MB	gsod_1958.tar

> 数据压缩输入txt文件准备

每年的数据文件是每个气象站收集回传的，那么会生成很多小数据文件，每年的数据需要进行合并压缩成一个大文件，提供Hadoop计算效率。

需要合并先hdfs的每个压缩文件的目录作为输入，下面分别为生成输入文本文件的命令和最终文本文件内容。

输入文本文件命令

```
#!/bin/bash
a=$1
rm ncdc_files.txt
hdfs dfs -rm /data/ncdc_files.txt

while [ $a -le $2 ]
do
        filename="/data/gsod/gsod_${a}.tar"
        echo -e "$filename" >> ncdc_files.txt
        a=`expr $a + 1`
done

hdfs dfs -put ncdc_files.txt /data/
```

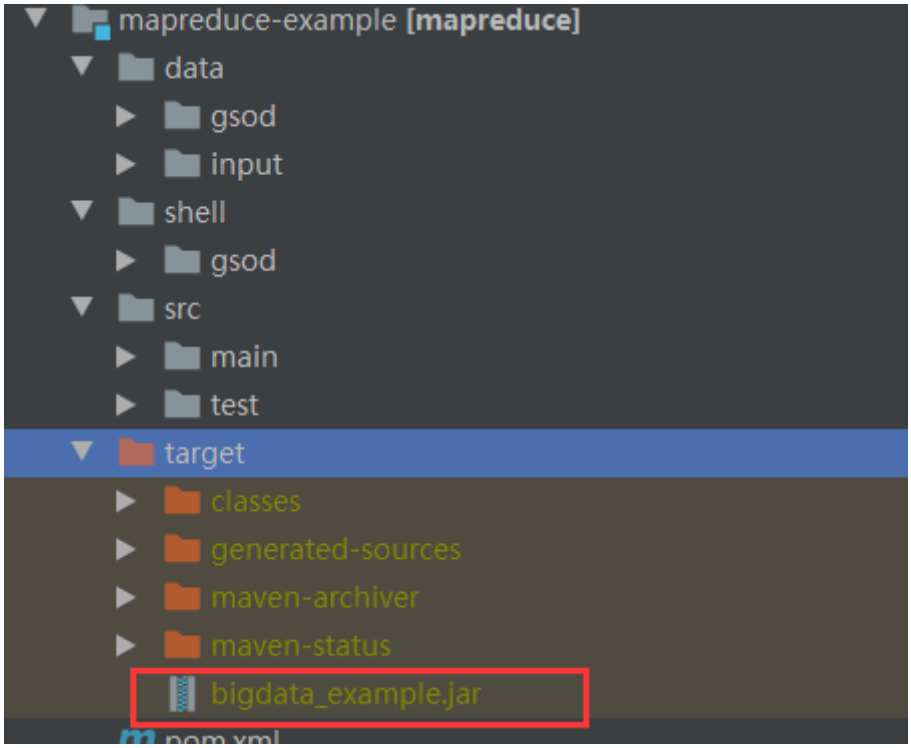
文件文件

```
/data/gsod/gsod_1949.tar
/data/gsod/gsod_1950.tar
/data/gsod/gsod_1951.tar
/data/gsod/gsod_1952.tar
/data/gsod/gsod_1953.tar
/data/gsod/gsod_1954.tar
/data/gsod/gsod_1955.tar
/data/gsod/gsod_1956.tar
/data/gsod/gsod_1957.tar
/data/gsod/gsod_1958.tar
/data/gsod/gsod_1959.tar
/data/gsod/gsod_1960.tar
```

4. 程序准备

4.1. 打包

工程DEMO直接Maven打包，在target生成jar包



4.2. 上传脚本和包

准备：
客户端服务器->这里的大数据集群TBDS的客户端服务器

步骤：

> tep1: ssh连接客户端服务器。

> tep2: 创建程序目录

```
mkdir -p /home/hadoop/gsod/jar
```

> tep3: xftp工具连上客户端环境

> tep4: 将工程下的shell脚本命令全部上传到/home/hadoop/gsod目录下

> tep5: 将打包生成的mapreduce.jar上传到/home/hadoop/gsod/jar目录下

> tep6: 修改shell脚本文件可执行权限

```
chmod +755 /home/hadoop/gsod/*.sh
```

4.3. hosts配置

理论上客户端配置文件都已经配置了hosts文件，但是如果集群下有扩展节点情况下需要替换hosts配置。

客户端应用服务器或客户端服务器需要hosts文件配置Hadoop集群的ip和hostname映射。

> 集群任何一台服务器下载/etc/hosts文件

> 在客户端服务器下替换其hosts配置。

5. MapReduce流

5.1. 需求说明

下载的气象数据每年的数据文件是很多压缩文件组合，需要优化处理每年的数据合并为一个大文件。

5.2. 技术选型

在HDFS上面不适合做压缩解压缩合并再压缩等操作，需要结合操作系统级别shell编程来实现，那么使用HadoopMR流处理进行本地Linux命令进行解压、打包、合并成一个大数据文件然后再压缩再上传到HDFS，需要进行进行shelll编程的进行Map计算，最后合并的数据合并到/data/gsod_all/目录下。

5.3. MR流计算

5.3.1. Map代码

map脚本：shell/gsod/load_ncdc_map.sh

```
#!/bin/bash
read offset hdfs_file
echo "$hdfs_file"
echo -e "$offset\t$hdfs_file"
echo "reporter:status:Retrieving $hdfs_file" >&2
# 从HDFS get文件到本地磁盘
hdfs dfs -get $hdfs_file .
# 创建本地文件
target=`basename $hdfs_file .tar`
mkdir $target

echo "reporter:status:Un-tarring $hdfs_file to $target" >&2
# 解压包到本地
tar xf `basename $hdfs_file` -C $target
# 本地解压每个压缩数据包下小文件并合并到一个大文件下
echo "reporter:status:Un-gzipping $target" >&2
for file in $target/*
do
        gunzip -c $file >> $target.all
        #echo "reporter:status:Processed $file" >&2
done
# 压缩数据文件并上传到HDFS合并后的输入目录（/data/gsod_all）下
echo "reporter:status:Gzipping $target and putting in HDFS" >&2
gzip -c $target.all | hdfs dfs -put - /data/gsod_all/$target.gz
# 删除本地文件
rm `basename $hdfs_file`
rm -r $target
rm $target.all
```

5.3.2. 命令脚本

执行命令脚本在工程的shell/merge_data_job.sh,具体命令如下：

```
#!/bin/bash
hadoop jar /opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0.p0.21/lib/hadoop-mapreduce/hadoop-stream
-D mapreduce.job.reduces=0 \
-D mapreduce.map.speculative=false \
-D mapreduce.task.timeout=12000000 \
-inputformat org.apache.hadoop.mapred.lib.NLineInputFormat \
-input /data/ncdc_files.txt \
-output /output/1 \
-mapper load_ncdc_map.sh \
-file load_ncdc_map.sh
```

命令说明：

- jar 包使用开源包hadoop-streaming.jar
- reduces 数为0即为不进行reduces计算
- 是否推导执行false即为关闭推导执行
- task超时时间：12000000，即为3.33小时
- 输入格式为NLineInputFormat
- -input 输入为ncdc_files.txt
- -output 输出目录/output/1
- mapper计算为load_ncdc_map.sh
- 文件为load_ncdc_map.sh

5.3.3. MR流执行

执行命令

```
# 合并气象数据
sh /home/hadoop/gsod/merge_data_job.sh
```

执行情况

执行命令

```
[root@hadoop01 ~]# sh /home/hadoop/gsod/merge_data_job.sh
19/10/21 16:33:18 WARN streaming.StreamJob: -file option is deprecated, please use generic option
packageJobJar: [/home/hadoop/gsod/load_ncdc_map.sh] [/opt/cloudera/parcels/CDH-5.15.0-1.cdh5.15.0
19/10/21 16:33:20 INFO mapred.FileInputFormat: Total input paths to process : 1
19/10/21 16:33:20 INFO mapreduce.JobSubmitter: number of splits:14
19/10/21 16:33:20 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1571638426016_0001
19/10/21 16:33:20 INFO impl.YarnClientImpl: Submitted application application_1571638426016_0001
19/10/21 16:33:20 INFO mapreduce.Job: The url to track the job: http://hadoop01:8088/proxy/applic
19/10/21 16:33:20 INFO mapreduce.Job: Running job: job_1571638426016_0001
19/10/21 16:33:28 INFO mapreduce.Job: Job job_1571638426016_0001 running in uber mode : false
19/10/21 16:33:28 INFO mapreduce.Job:  map 0% reduce 0%
19/10/21 16:34:02 INFO mapreduce.Job:  map 14% reduce 0%
19/10/21 16:34:03 INFO mapreduce.Job:  map 21% reduce 0%
19/10/21 16:34:07 INFO mapreduce.Job:  map 29% reduce 0%
19/10/21 16:34:09 INFO mapreduce.Job:  map 36% reduce 0%
19/10/21 16:34:41 INFO mapreduce.Job:  map 43% reduce 0%
19/10/21 16:34:43 INFO mapreduce.Job:  map 50% reduce 0%
19/10/21 16:34:51 INFO mapreduce.Job:  map 57% reduce 0%
19/10/21 16:34:53 INFO mapreduce.Job:  map 64% reduce 0%
19/10/21 16:35:01 INFO mapreduce.Job:  map 71% reduce 0%
19/10/21 16:35:25 INFO mapreduce.Job:  map 79% reduce 0%
19/10/21 16:35:33 INFO mapreduce.Job:  map 86% reduce 0%
19/10/21 16:35:42 INFO mapreduce.Job:  map 93% reduce 0%
19/10/21 16:35:43 INFO mapreduce.Job:  map 100% reduce 0%
19/10/21 16:35:44 INFO mapreduce.Job: Job job_1571638426016_0001 completed successfully
19/10/21 16:35:44 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=2174442
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
```

检查输出结果

```
# 查看合并数据
hdfs dfs -ls /output/1
hdfs dfs -cat /ouput/1/part-00000
```

输出结果

合并气象数据

hdfs dfs -ls /output/1

hdfs dfs -cat /ouput/1/part-00000

```
[root@hadoop01 ~]# hdfs dfs -ls /output/1
Found 15 items
-rw-r--r--   3 root supergroup          0 2019-10-21 16:35 /output/1/_SUCCESS
-rw-r--r--   3 root supergroup         54 2019-10-21 16:34 /output/1/part-00000
-rw-r--r--   3 root supergroup         54 2019-10-21 16:34 /output/1/part-00001
-rw-r--r--   3 root supergroup         54 2019-10-21 16:34 /output/1/part-00002
-rw-r--r--   3 root supergroup         55 2019-10-21 16:34 /output/1/part-00003
-rw-r--r--   3 root supergroup         55 2019-10-21 16:34 /output/1/part-00004
-rw-r--r--   3 root supergroup         55 2019-10-21 16:34 /output/1/part-00005
-rw-r--r--   3 root supergroup         55 2019-10-21 16:34 /output/1/part-00006
-rw-r--r--   3 root supergroup         55 2019-10-21 16:34 /output/1/part-00007
-rw-r--r--   3 root supergroup         55 2019-10-21 16:34 /output/1/part-00008
-rw-r--r--   3 root supergroup         55 2019-10-21 16:35 /output/1/part-00009
-rw-r--r--   3 root supergroup         55 2019-10-21 16:35 /output/1/part-00010
-rw-r--r--   3 root supergroup         55 2019-10-21 16:35 /output/1/part-00011
-rw-r--r--   3 root supergroup         55 2019-10-21 16:35 /output/1/part-00012
-rw-r--r--   3 root supergroup         53 2019-10-21 16:35 /output/1/part-00013
[root@hadoop01 ~]#
[root@hadoop01 ~]#
[root@hadoop01 ~]#
[root@hadoop01 ~]# hdfs dfs -cat /output/1/part-00000
/data/gsod/gsod_1950.tar
25      /data/gsod/gsod_1950.tar
```

查看HDFS数据合并后的数据文件

HDFS数据合并后的数据文件

/data/gsod all

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	yarn	supergroup	9.35 MB	Mon Oct 21 16:35:23 +0800 2019	3	128 MB	gsod_1949.gz
-rw-r--r--	yarn	supergroup	10.62 MB	Mon Oct 21 16:34:00 +0800 2019	3	128 MB	gsod_1950.gz
-rw-r--r--	yarn	supergroup	10.81 MB	Mon Oct 21 16:33:59 +0800 2019	3	128 MB	gsod_1951.gz
-rw-r--r--	yarn	supergroup	12.07 MB	Mon Oct 21 16:34:00 +0800 2019	3	128 MB	gsod_1952.gz
-rw-r--r--	yarn	supergroup	12.95 MB	Mon Oct 21 16:34:05 +0800 2019	3	128 MB	gsod_1953.gz
-rw-r--r--	yarn	supergroup	13.73 MB	Mon Oct 21 16:34:06 +0800 2019	3	128 MB	gsod_1954.gz
-rw-r--r--	yarn	supergroup	13.19 MB	Mon Oct 21 16:34:41 +0800 2019	3	128 MB	gsod_1955.gz
-rw-r--r--	yarn	supergroup	13.58 MB	Mon Oct 21 16:34:39 +0800 2019	3	128 MB	gsod_1956.gz
-rw-r--r--	yarn	supergroup	17.11 MB	Mon Oct 21 16:34:48 +0800 2019	3	128 MB	gsod_1957.gz
-rw-r--r--	yarn	supergroup	18.41 MB	Mon Oct 21 16:34:51 +0800 2019	3	128 MB	gsod_1958.gz

6. MapReduce编程

基于全球气象合并后的数据求每年的最高气温，通过本示例了解MR基础编程实现、数据流程、配置作业、shuffle、排序、运行方式、yarn作业管理等。

6.1. 示例需求

基于全球的气象数据进行数据挖掘，目标需要查询出每年的最高气温分别为多少。

需要求出最高去温那么得从文本数据根据数据结构解析到气温记录，得到每年的全部气温数据，然后进行数学计算求最大值得到最高气温。

6.2. MapReduce编程说明

MapRedcue编程有两个阶段为Mapper阶段和Reduce阶段，每个阶段都以Kay-Value方式作为输入和输出，其类型都由编程人员自己定但是要满足一定的规范，编程人员就是自己去需要编写实现Mapper和Reduce两个函数。编程人员还需要编写Driver去配置参数和创建Job并启动MapReduce Job。

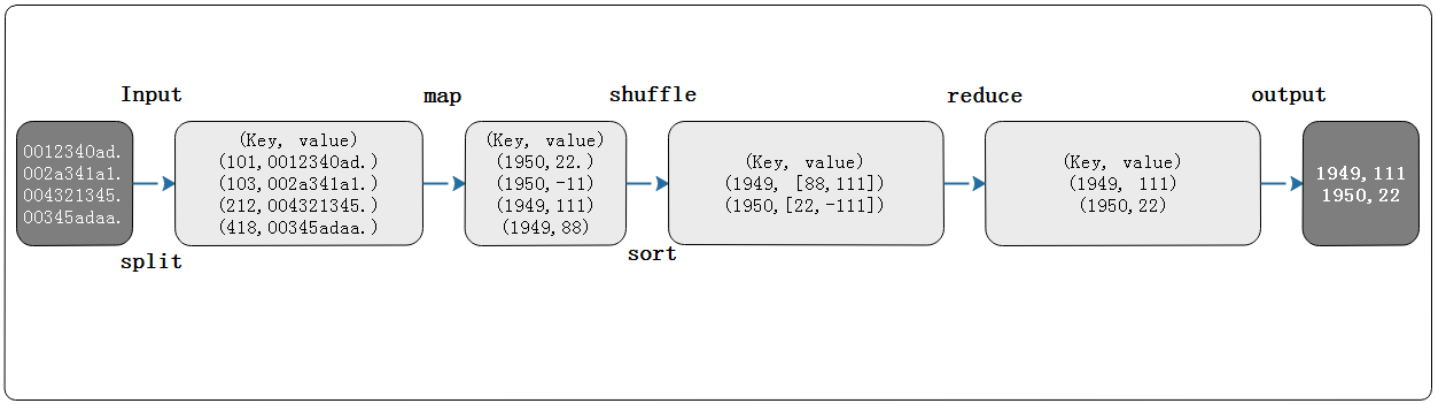
在Reduce输入会根据Map输入的值进行合并，输入类型为[V, Itable[V]]也就是相同的Key会进行合并，那么这很适合获取每年的气温数据，也在Maper阶段就是Key作为年，气温作为V值输出，在Reduce过程就可以得到每年的全部气温数据，然后在迭代进行求最大值即可。

6.3. MapReduce数据流程分析

根据需求和源数据进行数据流程分析，mr的计算每个都是通过Key-Value进行输出，注意每个key-value都需要定义Hadoop系列化类型：

- input分片：hdfs文本文件作为输入时候，输出应该key为偏移量，value一行文本数据值
- map阶段：输出key为年份，value值为气温值
- suffle阶段： 输出为key年份，value为全部年份的气温的迭代器
- reduce阶段：计算每年的最高气温后，key为年份，value为最高气温
- output阶段：每年的气温和对应最高气温通过文本输出到hdfs

数据流程图



6.4. Mapper编程

类：com.tencent.bigdata.example.mapreduce.map.MaxTemperatureMapper

代码核心是继承org.apache.hadoop.mapreduce.Mapper确定输入输出、重写map方法，读取每一行的数据，每行的数据然后获取年份及对应的气温数据，作为输出，中间还进行一些数据过滤和异常数据判断等。

```
public class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {
    // One or more space char
    private static final String FIELD_SPLIT = "\\s+";
    // Abnormal temperature
    private static final double MISS_TEMP = 9999.9;
    public enum Counter {
        ROWS_TOTAL, ERROR_NUMBER //总数据量，异常数据量
    }
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        //Filter first line
        if(line.startsWith("STN---")) return;
        // rows total counter
        context.getCounter(MaxTemperatureMapper.Counter.ROWS_TOTAL).increment(1);
        // get year
        String year = line.split(FIELD_SPLIT)[2].substring(0, 4);
        double airTemperature;
        // get max temperature
        String airTemperatureStr = line.split(FIELD_SPLIT)[17];
        // delete *
        airTemperatureStr = airTemperatureStr.replace( target: "*",  replacement: "");
        airTemperature = Double.parseDouble(airTemperatureStr);
        //Filter miss temp
        if(airTemperature == MISS_TEMP){
            context.getCounter(Counter.ERROR_NUMBER).increment(1);
            return;
        }
        System.out.println(airTemperature);

        //output
        context.write(new Text(year), new DoubleWritable(airTemperature));
    }
```


6.5. Reduce编程

示例代码

类：com.tencent.bigdata.example.mapreduce.reduce.MaxTemperatureReduce

```java
/**
 * @author liulv
 *
 * @description Reduce
 */
public class MaxTemperatureReduce extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
            throws IOException, InterruptedException {

        double maxValue = Double.MIN_EXPONENT;
        for(DoubleWritable value : values){
            maxValue = Math.max(maxValue, value.get());
            System.out.println(maxValue);
        }

        System.out.println("================================================");
        System.out.println(maxValue);
        context.write(key, new DoubleWritable(maxValue));
    }
}
```

31%
2.6K/s

说明

- Reduce开发需要继承org.apache.hadoop.mapreduce.Reduce
- 重写reduce方法，输入类型必须与Map类的输出需要完全一致
- Reduce获取输入数据为suffel阶段也就是mapper阶段向reduce传递数据过程
- 输入值为同一年只一个key值，value为同一年的全部最高气温的迭代器
- 通过Math.max函数读取最高气温
- 输出值key为年份，输出value为计算后的最高气温

7. Driver类开发

实例代码：com.centent.bigdata.example.mapreduce.driver.MaxTemperatureDriver

```java
@SuppressWarnings( "DuplicatedCode" )
public class MaxTemperatureDriver {

    public static void main(String[] args) throws IOException, ClassNotFoundException, Int
        if(args.length != 2){
            System.err.println("Usage: MaxTemperature <input path> <output path>");
            System.exit( status: -1);
        }
        //Initialize the job instance
        Job job = Job.getInstance();
        job.setJarByClass(MaxTemperatureDriver.class);
        job.setJobName("Max Temperature");
        // Set the reduce number of
        job.setNumReduceTasks(1);
        //input
        FileInputFormat.addInputPath(job, new Path(args[0]));
        //output
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //Set the mapper class for the job.
        job.setMapperClass(MaxTemperatureMapper.class);
        //Set the reduce class for the job.
        job.setReducerClass(MaxTemperatureReduce.class);
        //Set the InputFormat for the job.
        job.setInputFormatClass(TextInputFormat.class);
        //Set the output's kay and value for the job.
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);
        //Submit the job to the cluster and wait for it to finish.
        int status = job.waitForCompletion( verbose: true) ? 0 : 1;
        long rows_total;
        if(job.isSuccessful()){
```

说明：配置和启动MapReduce作业

- 初始化 Job类并设置jar包，设置作业名和Jar包
- 设置reduce阶段task数
- 设置输入和输出格式
- 设置设置输入和输出目录
- 设置Mapper和Reduce类
- 启动作业
- 作业成功后通过计数器获取总数据量和异常数据量

8. 运行MapReduce

运行提交作业最终是提交到Yarn上面执行，提交方式有本地提交和远程提交。

> - 远程提交也是就是开发环境IDEA直接进行提交作业
> - 本地提交直接基于客户端服务器进行提交作业

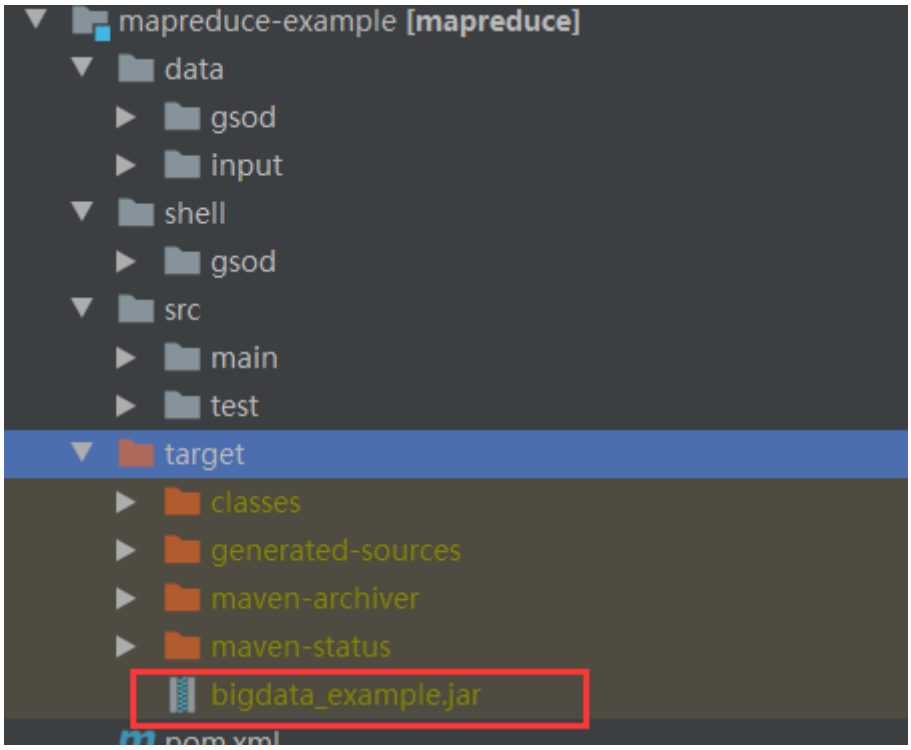
不管是本地提交还是远程提交都是一个Java进场都是相当于客户端运行程序。Yarn运行作业也就是服务端进行处理，如果不进行计数打印或者其他判断，基本客户端提交完作业使命就是算完成了。

我们知道Hadoop是基于Java开发的，既然都是客户端方式那么打成java Jar包是必不少的，MapReduce作业运行在Yarn上面不过是把本地包上传到集群而已，还有一些参数，最终根据提交的Jar包的指定方法调用HDFS临时的Jar包依赖进行计算处理。当然暂时这些过程不用太过关心。

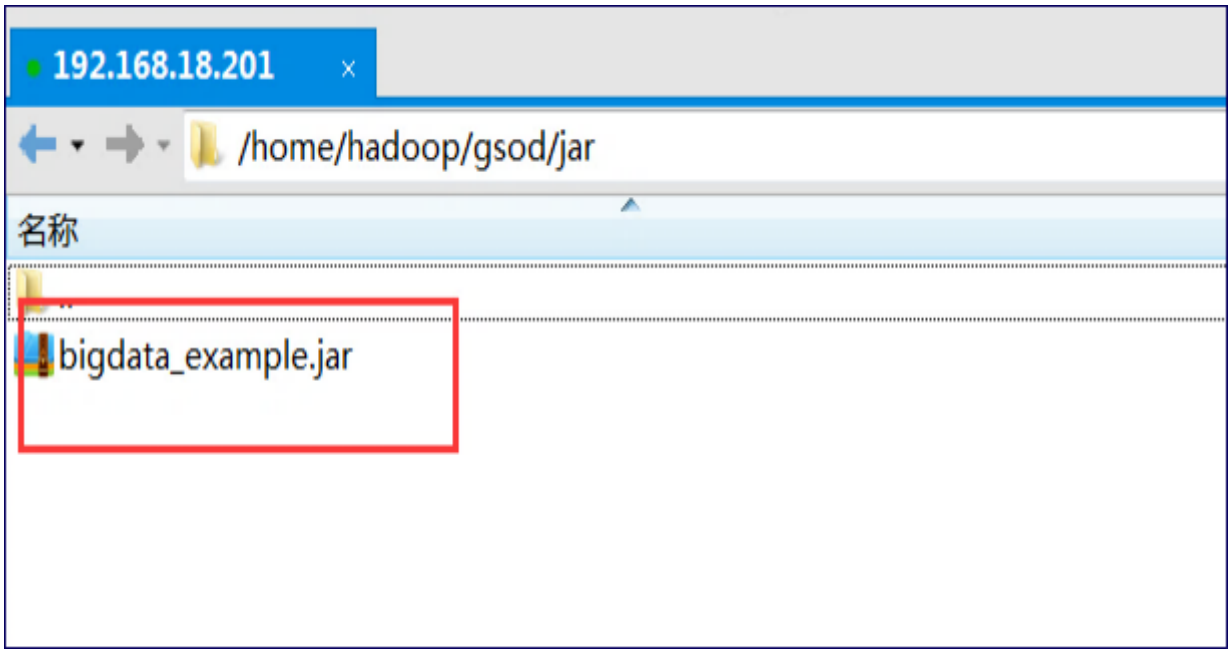
8.1. Maven打包

全面准备过程已经打包过了，但是程序有改动调整，那么需要重新打包上传。

直接通过Maven package进行打包即可，打包最终生成的Jar包在工程的target目录下，Jar包名为bigdata_example.jar。

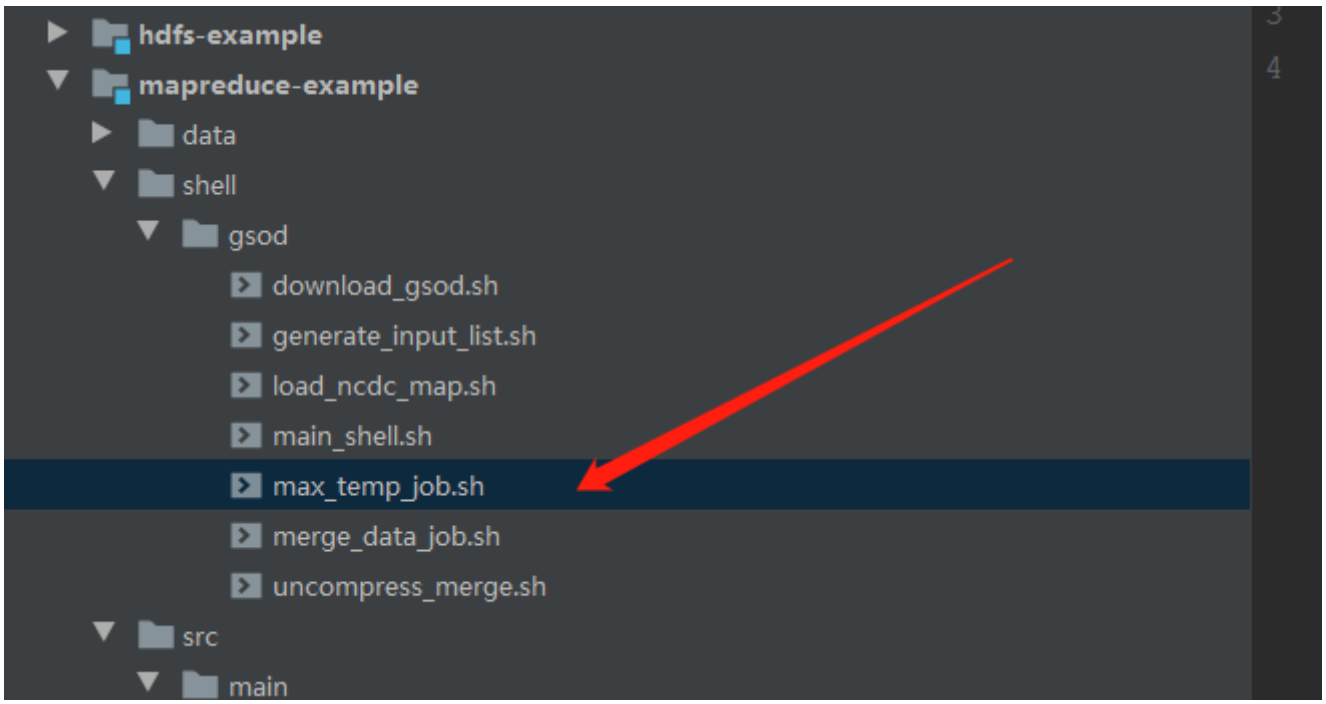


这里示例演示是本地提交模式，直接上传客户端服务器下的/home/hadoop/gsod/jar目录下。



8.2. 提交运行作业

也就是通过Hadoop jar方式在客户端进行作业提交到Yarn上面执行，shell脚本在工程的shell/gsod/max_temp_job.sh




```
hadoop jar /home/hadoop/gsod/jar/bigdata_example.jar \
com.sinobest.bigdata.example.mapreduce.driver.MaxTemperatureDriver \
/data/gsod_all/ \
/output/2
```

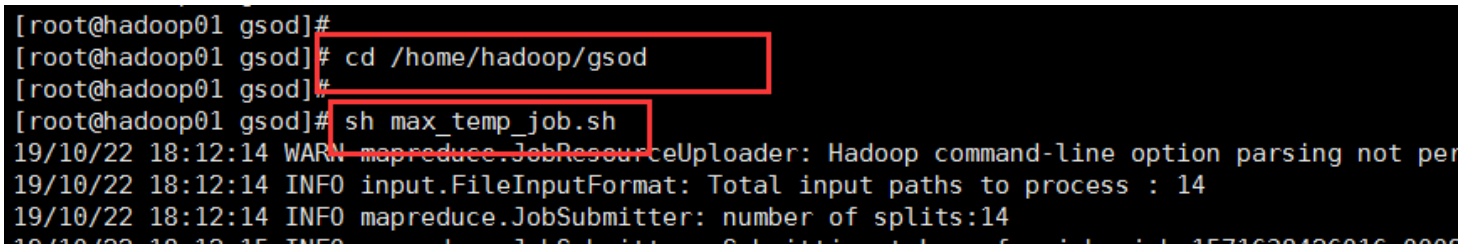
注意：最后行参数根据实际进行调整。

> 执行命令

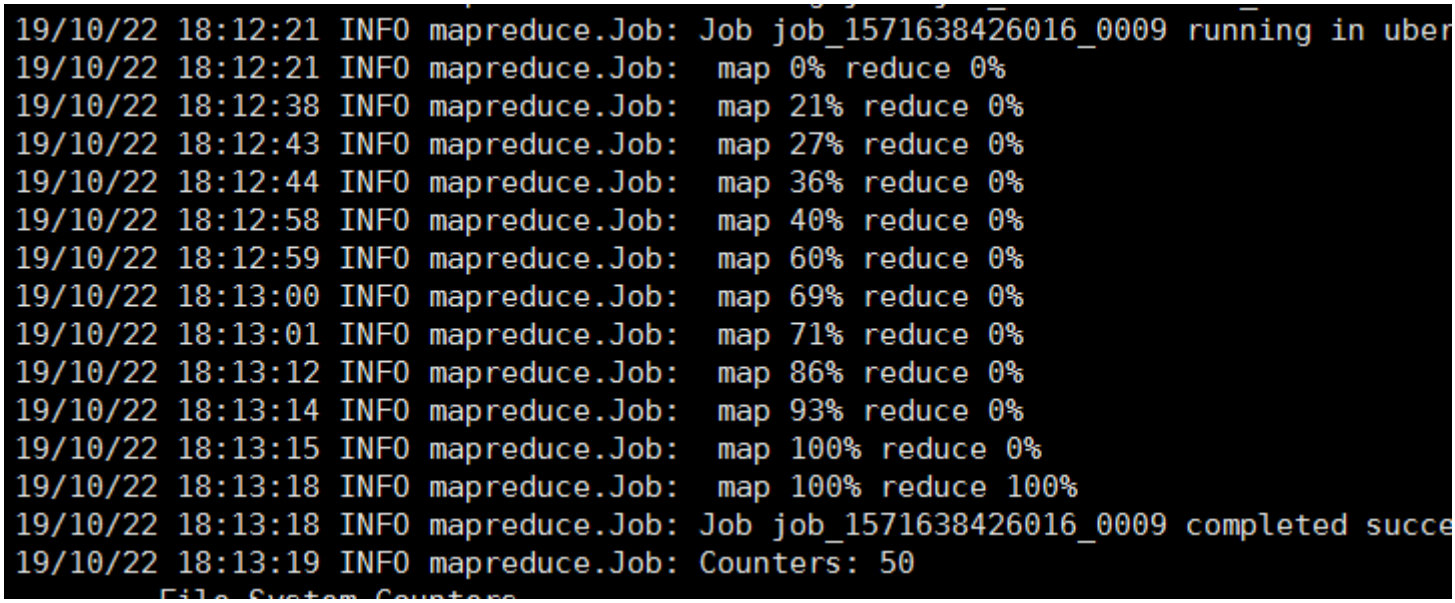
```
cd /home/hadoop/psod
./max_temp_job.sh
```

> 也可以通过sh命令方式执行

```
sh /home/hadoop/psod/max_temp_job.sh
```

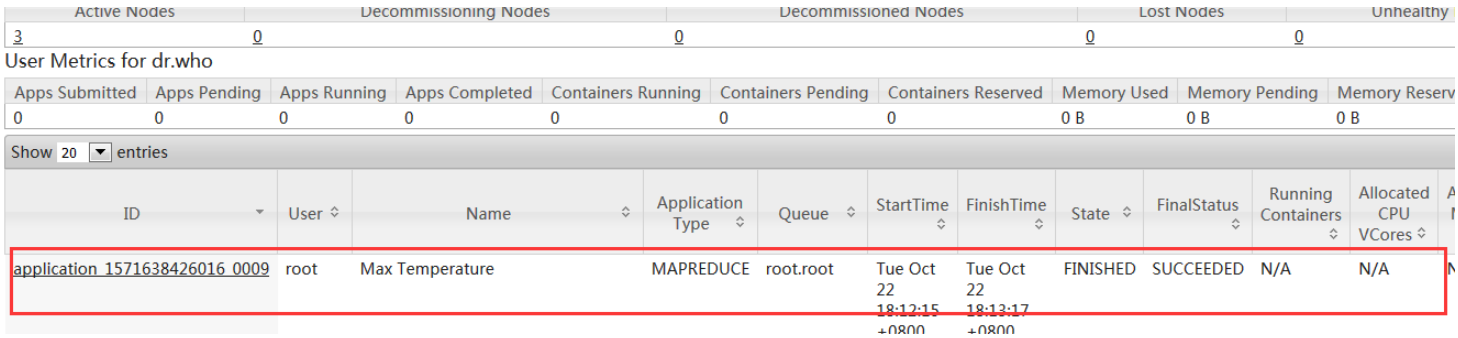


> 执行过程



8.3. 运行结果

运行作业情况可以在Yarn上面进行查看，如果maper或reduce过程进行了日志打印，那么也必须在Yarn的Map或Reduce Task运行日志中查看。



Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy
3	0	0	0	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserv
0	0	0	0	0	0	0	0 B	0 B	0 B

Show 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores
application_1571638426016_0009	root	Max Temperature	MAPREDUCE	root.root	Tue Oct 22 18:12:15 +0800	Tue Oct 22 18:13:17 +0800	FINISHED	SUCCEEDED	N/A	N/A

输出数据在程序中我们自己写入到hdfs输出/output/2目录下，直接hdfs命令去查看计算结果。

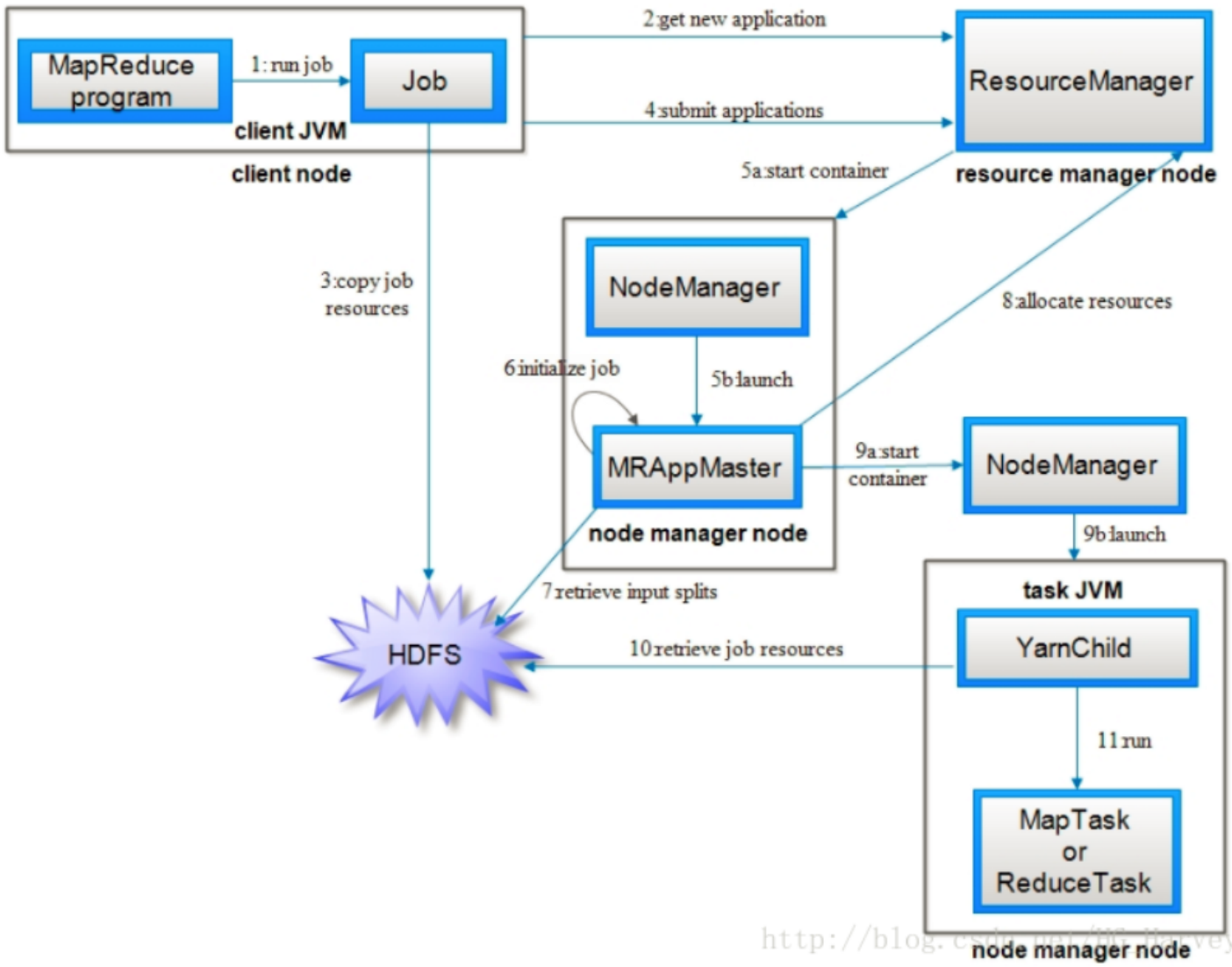
```
[root@hadoop01 gsod]#
[root@hadoop01 gsod]# hadoop fs -ls /output/2/
Found 2 items
-rw-r--r--   3 root supergroup          0 2019-10-22 18:20 /output/2/_SUCCESS
-rw-r--r--   3 root supergroup        154 2019-10-22 18:20 /output/2/part-r-00000
[root@hadoop01 gsod]#
[root@hadoop01 gsod]#
[root@hadoop01 gsod]# hadoop fs -cat /output/2/part-r-00000
1949    120.0
1950    120.0
1951    127.0
1952    123.1
1953    120.0
1954    120.0
1955    122.0
1956    119.3
1957    120.0
1958    120.9
1959    120.0
1960    120.2
1961    127.0
1962    120.9
[root@hadoop01 gsod]#
```

9. MapReduce机制和性能

基于MR运行机制并在上面示例进行性能优化处理，包含压缩、task数调整，分片split设置、Combiner、task内存，reduce延迟加载、suffle参数。

9.1. MapReduce工作流程

工作流程图



流程说明

1).客户端通过submit()或waitForCompletion()方法提交作业

2).从资源管理器获取作业ID,并检查作业检查本次作业是否可执行

3).可执行情况将作业运行所需要的jar包、配置文件、分片信息等上传到HDFS

4).调用submitApplication()方法提交作业

5).请求传递给调度器，调度器分配容器，资源管理器在节点管理器管理的容器中启动应用程序进程

6). MRAppMaster对作业进行初始化，监控任务的进度和完成情况。

7). MRAppMaster接受来自hdfs在客户端计算的输入分片

8). 每一个分片启动一个map task，reduce task根据配置（mapreduc.job.reduce），全部任务向资源管理器申请容器，并分配内存，默认情况每个任务分配1024M

9). application master 通过节点管理器通讯启动容器。

10). 运行前资源本地化，从HDFS获取作业配置、Jar包、缓存文件等。

11). 运行Map任务和Reduce任务

9.2. 性能优化思路

MapReduce的机制最重要就是其实就进行数据Split创建Map并非数以及读取读取后的程序效率，以及Maper输出到Reduce过程的也就是Shuffle过程，Reduce程序计算逻辑，以及整个数据传输过程的优化处理。

优化路线

- Mapper并非数
  决定因素为文件数以及文件大小、分片大小，可设置分片大小，合并文件分片
- 集群硬件、配置优化
- Reduce并非数
  通过Job的setNumReduceTasks设置
- 中间值压缩
  中间值压缩也是map输出启动压缩
- 自定义Hadoop系列
  使用自定义的Writable，则必须实现RawComparator
- combiner
  充分应用combiner来减少shuffle传输的数据量
- 调整shuffle
  配置调整shuffle过程的参数，提升性能
- 程序调优