**大数据技术实践**

利用MapReduce进行数据的查询计算

1. **引言**

本次作业主要完成在Hadoop平台上利用MapReduce完成数据的查询计算，分别是：文本词频统计、文本倒排索引、数据去重、数据表查询四个计算。

1. **技术介绍**

**MapReduce**

1. 原理：Map代表映射，Reduce代表归约。 是Hadoop进行分布式数据计算的模式，即将文件分发Map到各节点进行计算，然后在将各节点计算结果汇总Reduce成最终结果。
2. 通用过程：

****Map端：****

**1.Map**输出的结果会暂且放在一个环形内存缓冲区中，当该缓冲区快要溢出时会在本地文件系统中创建一个溢出文件，将该缓冲区中的数据写入这个文件。

2.在写入磁盘之前，线程首先根据reduce任务的数目将数据划分为相同数目的分区，就是一个reduce任务对应一个分区的数据。

3.当map任务输出最后一个记录时，可能会有很多的溢出文件，这时需要将这些文件合并。

4.将分区中的数据拷贝给相对应的reduce任务。

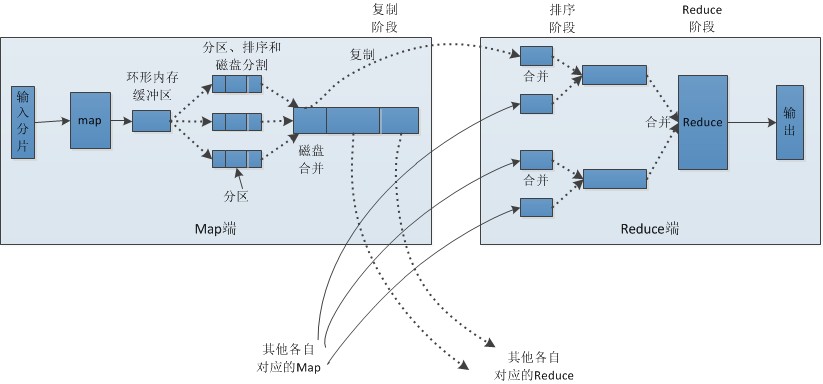
****Reduce端：****

**1**.**Reduce**会接收到不同map任务传来的数据，并且每个map传来的数据都是 有序的。如果reduce端接受的数据量相当小，则直接存储在内存中。

2.随着溢写文件的增多，后台线程会将它们合并成一个更大的有序的文件，这样做是为了给后面的合并节省时间。

3.合并的过程中会产生许多的中间文件（写入磁盘了），但MapReduce会让写入磁盘的数据尽可能地少，并且最后一次合并的结果并没有写入磁盘，而是直接输入到reduce函数。

Visio展示工作流程示意图：



1. **数据描述**

数据名称：亚马逊的客户评论信息

Amazon Commerce reviews set Data Set

数据类型：文本类型

Multivariate, Text, Domain-Theory

数据特征：

来自Amazon Commerce网站中的客户评论，以鉴定作者身份确定了50个最活跃的用户。

数据来源：

<http://archive.ics.uci.edu/ml/datasets/Amazon+Commerce+reviews+set>

1. **实践过程**

* 文本词频统计

（1）**具体过程**：对于给定的文档，确定每个单词存在于某个文档，同时在文档中出现的次数Map端对文件统计每个单词出现的次数，输出类似<{hadoop,file1},2>

（2）**代码**：

##### **Mapper：WordCountMapper**

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

// 通过分隔符分割单词

String[] words = value.toString().split("\t");

// 遍历单词

for (String word: words) {

// 写入到context中

context.write(new Text(word), new IntWritable(1));

}

}

}

##### **Reducer：**

##### Reduce 将一组中间值转化成共享一个key，value合并成一组较小的值

# 从文件中读取的单词

(hello,1) (world,1)

(hello,1) (world,1)

(hello,1) (world,1)

(welcome,1)

# map的输出到reduce端，是按照相同的key分发到一个reduce上去执行

reduce1: (hello,1) (hello,1) (hello,1) ===> (hello, <1,1,1>)

reduce2: (world,1) (world,1) (world,1) ===> (world, <1,1,1>)

reduce3: (welcome,1) ===> (welcome, <1>)

**自定义词频Reducer：WordCountReducer**

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,

InterruptedException {

int count = 0;

Iterator<IntWritable> iterator = values.iterator();

// <1,1,1>

while (iterator.hasNext()) {

IntWritable value = iterator.next();

count += value.get();

}

context.write(key, new IntWritable(count));

}

}

**词频统计-Driver**

public class WordCountApp {

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

System.setProperty("HADOOP\_USER\_NAME", "root");

// 设置HDFS的Configuration

Configuration configuration = new Configuration();

configuration.set("fs.defaultFS", "hdfs://139.129.240.xxx:8020");

configuration.set("dfs.client.use.datanode.hostname", "true");

configuration.set("dfs.replication", "1");

// 创建一个job

Job job = Job.getInstance(configuration);

// 设置Job对应的参数：主类

job.setJarByClass(WordCountApp.class);

// 设置Job对应的参数：设置自定义的Mapper和Reducer处理类

job.setMapperClass(WordCountMapper.class);

job.setReducerClass(WordCountReducer.class);

// 设置Job对应的参数：Mapper输出key和value的类型

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

// 设置Job对应的参数：Reducer输出key和value的类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

// 设置Job对应的参数：设置输入/输出路径

FileInputFormat.setInputPaths(job, new Path("/hdfsapi/test/second/words.txt"));

FileOutputFormat.setOutputPath(job, new Path("/wordcount/output"));

// 提交job

job.waitForCompletion(true);

}

}

**本地测试**

设置Job对应的参数：设置输入/输出路径，设置成本地路径即可

public class WordCountLocalFileApp {

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

// 创建一个job

Job job = Job.getInstance();

// 设置Job对应的参数：主类

job.setJarByClass(WordCountLocalFileApp.class);

// 设置Job对应的参数：设置自定义的Mapper和Reducer处理类

job.setMapperClass(WordCountMapper.class);

job.setReducerClass(WordCountReducer.class);

// 设置Job对应的参数：Mapper输出key和value的类型

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

// 设置Job对应的参数：Reducer输出key和value的类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

// 设置Job对应的参数：设置输入/输出路径

FileInputFormat.setInputPaths(job,new Path("/Users/xxxx/IdeaProjects/bigdata/hadoop-mapreduce/src/main/resources/words.txt"));

FileOutputFormat.setOutputPath(job,new Path("/Users/xxxx/IdeaProjects/bigdata/hadoop-mapreduce/src/main/resources/output"));

// 提交job

job.waitForCompletion(true);

}

}

（3）**运行**：

1. 启动Hadoop

    切换到Hadoop安装目录下的sbin目录下执行./start-all.sh命令

2. 在集群中创建目录inputdata\_w

    hdfs dfs -mkdir /inputdata\_w

3. 将Hadoop安装目录下的LICENSE.txt、README.txt、NOTICE.txt文件上传到集群

    hdfs dfs -put ../LICENSE.txt /inputdata\_w

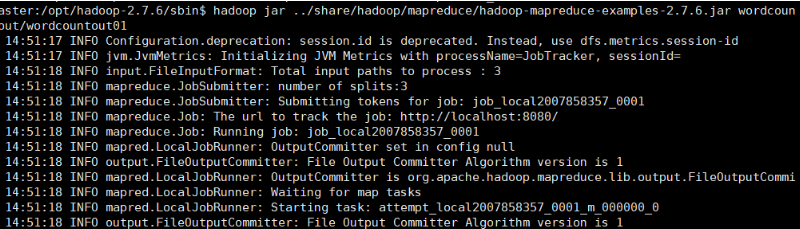
    hdfs dfs -put ../README.txt /inputdata\_w

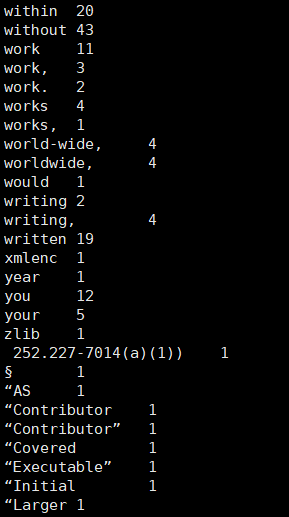
    hdfs dfs -put ../NOTICE.txt /inputdata\_w

1. 使用hadoop-mapreduce-examples-2.7.6.jar对上传的数据进行词频统计

5.查看统计结果：hdfs dfs -cat /output/wordcountout01/part-r-00000

**（4）结果截图**





* 文本倒排索引

1. **具体过程：**

索引中的每一项都包括一个属性值和具有该属性值的各记录的地址，正常的索引思想是先将各个文件中的单词列出来，然后进行查找，而单排索引是将查找的关键字以格式（关键字，<文件名，出现次数>）的列表进行查询

1. **代码**

**inverseIndexMap类：**

package com.testMapReduce8\_1;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;import org.apache.hadoop.io.Text;

Importorg.apache.hadoop.mapreduce.Mapper;import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class inverseIndexMap1 extends Mapper<LongWritable, Text, Text, LongWritable>{

@Override

protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, LongWritable>.Context context)

throws IOException, InterruptedException {

String [] words = value.toString().split(" ");

String fileName = ((FileSplit)context.getInputSplit()).getPath().getName();

for (String word : words) {

if (word !=null && word.length()!=0) {

context.write(new Text(word + "\t" + fileName), new LongWritable(1));

}

}

}

}

**inverseIndexReducer类：**

package com.testMapReduce8\_1;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;import org.apache.hadoop.mapreduce.Reducer;

public class inverseIndexReducer extends Reducer<Text, LongWritable, Text, LongWritable>{

@Override

protected void reduce(Text key, Iterable<LongWritable> values, Reducer<Text, LongWritable, Text, LongWritable>.Context context)

throws IOException, InterruptedException {

long sum = 0;

for (LongWritable value : values) {

sum += value.get();

}

context.write(key, new LongWritable(sum));

}

}

**InverseIndex类：**

package edu.jianwei.hadoop.mr.ii;

import java.io.IOException;

public class InverseIndex {

static class IndexMapper extends Mapper<LongWritable, Text, Text, Text> {

private final Text k = new Text();

private final Text v = new Text();

@Override

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

String[] words = line.split(" ");

FileSplit inputSplit = (FileSplit) context.getInputSplit();

String path = inputSplit.getPath().toString();

for (String word : words) {

k.set(word + "->" + path);

v.set("1");

context.write(k, v);

}

}

}

static class IndexCombiner extends Reducer<Text, Text, Text, Text> {

private final Text key = new Text();

private final Text value = new Text();

@Override

protected void reduce(Text k, Iterable<Text> v2s, Context context)

throws IOException, InterruptedException {

String line = k.toString();

String[] wordAndpath = line.split("->");

key.set(wordAndpath[0]);

int counter = 0;

for (Text v : v2s) {

counter += Integer.parseInt(v.toString());

}

value.set(wordAndpath[1] + "->" + counter);

context.write(key, value);

}

}

static class IndexReducer extends Reducer<Text, Text, Text, Text> {

private final Text v = new Text();

@Override

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

String value = "";

for (Text v : values) {

value += v.toString() + " ";

}

v.set(value);

context.write(key, v);

}

}

public static void main(String[] args) throws IOException,

InterruptedException, ClassNotFoundException {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf);

job.setJarByClass(InverseIndex.class);

job.setMapperClass(IndexMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

FileInputFormat.setInputPaths(job, new Path(args[0]));

job.setCombinerClass(IndexCombiner.class);

job.setReducerClass(IndexReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);

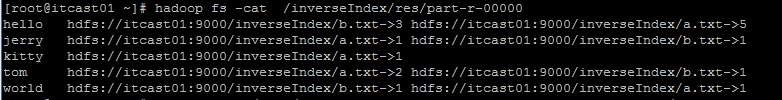
}

}

1. **运行：**

 hadoop jar  /root/ii.jar edu.jianwei.hadoop.mr.ii.InverseIndex  /inverseIndex  /inverseIndex/res

1. **结果截图：**



* 数据去重

1. **具体过程：**

对输入的多个文件进行合并，并剔除其中重复的内容，去重后的内容输出到一个文件中。

1. **代码：**

package com.javacore.hadoop;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.Job.

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import java.io.IOException;

/\*\*

\* Created by bee on 17/3/25.

\*/

public class FileMerge {

public static class Map extends Mapper<Object, Text, Text, Text> {

private static Text text = new Text();

public void map(Object key, Text value, Context content) throws IOException, InterruptedException {

text = value;

content.write(text, new Text(""));

}

}

public static class Reduce extends Reducer<Text, Text, Text, Text> {

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

context.write(key, new Text(""));

}

}

public static void main(String[] args) throws Exception {

// delete output directory

FileUtil.deleteDir("output");

Configuration conf = new Configuration();

conf.set("fs.defaultFS", "hdfs://localhost:9000");

String[] otherArgs = new String[]{"input/filemerge/f\*.txt",

"output"};

if (otherArgs.length != 2) {

System.err.println("Usage:Merge and duplicate removal <in> <out>");

System.exit(2);

}

Job job = Job.getInstance();

job.setJarByClass(FileMerge.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));

FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

1. **运行:**

/usr/jdk1.7.0\_25/bin/javac Dedup.java

/usr/jdk1.7.0\_25/bin/jar cvf xx.jar Dedup\*class

$ hadoop jar xx.jar Dedup

查看结果：$ hadoop fs -cat /user/hadoop/dedup\_out/part-r-00000

**（4）结果截图**

输入：客户的一些信息  
张三 男 20 备注1  
张三 男 20 备注2  
张三 女 20 备注1  
李四 男 21 备注1  
李四 男 21 备注3

输出： Person [name=张三, sex=女, age=20, remark=备注1]

Person [name=张三, sex=男, age=20, remark=备注2]

Person [name=李四, sex=男, age=21, remark=备注3]

* 数据表查询

1. **具体过程：**

这里直接用MapReduce操作Mysql数据库，创建数据表，进行数据查询。

1. **代码：**

**自定义Recoder：CourceRecord**

public class CourceRecord implements Writable, DBWritable {  
  int id;  
  String name;  
  int count;  
  int no;  
    
//从in流反序列化出内容  
  @Override  
  public void readFields(DataInput in) throws IOException {  
    this.id = in.readInt();  
    this.name = Text.readString(in);  
    this.count = in.readInt();  
    this.no = in.readInt();  
  }  
  //将内容序列化到out流  
  @Override  
  public void write(DataOutput out) throws IOException {  
    out.writeInt(this.id);  
    Text.writeString(out, this.name);  
    out.writeInt(this.count);  
    out.writeInt(this.no);  
  }  
  @Override  
  public void readFields(ResultSet result) throws SQLException {  
    this.id = result.getInt(1);  
    this.name = result.getString(2);  
    this.count = result.getInt(3);  
    this.no = result.getInt(4);  
  }  
  @Override  
  public void write(PreparedStatement stmt) throws SQLException {  
    stmt.setInt(1, this.id);  
    stmt.setString(2, this.name);  
    stmt.setInt(3, this.count);  
    stmt.setInt(4, this.no);  
  }  
  @Override  
  public String toString() {  
    return new String(this.name + " " + this.count + " " + this.no);  
  }  
}

**map类**

public class DBMapper extends MapReduceBase implements  
    Mapper<LongWritable, CourceRecord, LongWritable, Text> {  
  public void map(LongWritable key, CourceRecord value,  
      OutputCollector<LongWritable, Text> collector, Reporter reporter)  
      throws IOException {  
    collector.collect(new LongWritable(value.id),  
    new Text(value.toString()));  
  }  
}

**driver驱动类DB**

public class DB {  
  public static void main(String[] args) throws IOException {  
    String[] argc={"jdbc:mysql://localhost/test","root", "111111"};   
    
    try {  
      JobConf conf = new JobConf(DB.class);  
      Class.forName("com.mysql.jdbc.Driver");  
      DBConfiguration.configureDB(conf, "com.mysql.jdbc.Driver",argc[0], argc[1], argc[2]);  
      String[] fields = { "id", "name", "count", "no" };  
      DBInputFormat.setInput(conf, CourceRecord.class, "tb", null,"id", fields);  
        
      conf.setInputFormat(DBInputFormat.class);  
      conf.setOutputKeyClass(LongWritable.class);  
      conf.setOutputValueClass(Text.class);  
    
        
      Path path = new Path("DBOUTPUT");  
      FileOutputFormat.setOutputPath(conf, path);  
        
      conf.setMapperClass(DBMapper.class);  
      conf.setReducerClass(IdentityReducer.class);

      //如果文件存在则删除  
      FileSystem hdfs = path.getFileSystem(conf);  
      if (hdfs.exists(path)) {  
        hdfs.delete(path, true);  
      }  
        
      JobClient.runJob(conf);  
    }  
    catch(ClassNotFoundException e) {  
        System.err.println("mysql.jdbc.Driver not found");    
    }   
  }

1. **运行：**

在DBOUTPUT目录下生成一个part-00000的文件，其内容为：

17 hello 28 17

18 [Hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库) 28 18

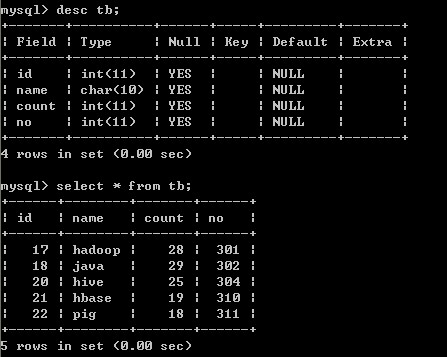
19 [Java](http://lib.csdn.net/base/java" \o "Java 知识库)28 19

20 [Hive](http://lib.csdn.net/base/hive" \o "Hive知识库) 16 20

21 [Hbase](http://lib.csdn.net/base/hbase" \o "Hbase知识库) 18 21

22 pig 18 22

1. **结果截图：**

****

1. **总结**

在完成本次大作业的过程中，我从MapReduce的工作原理，任务的执行过程等理论知识入手，再运用MapReduce进行了一系列的数据处理和查询，实现了四个功能：文本词频统计、文本倒排索引、数据去重、数据表查询；对hadoop下的MapReduce有了一个全面的认识。完成上机后，我分别对MapReduce编程核心思想、MapReduce进程、MapReduce编程规范进行了一些总结和归纳。我非常感激本次的大作业学习过程，因为在很大程度上激发了对大数据技术学习的热情，相信以后会继续对Hadoop下一些其他组件的学习和研究。