《大规模分布式系统》实验报告

——Hbase 操作

姓名：刘佰川 专业：计算机科学与技术（数据科学方向）学号：16307130214

0. 实验环境

VMware+Ubuntu18.04+Hadoop2.7.7+Java1.8.0_201+Hbase2.0.5

1. 实验要求

I. 用 Mapreduce 编程，实现两个表 N1 与 D1 的自然连接操作

II. 连接后的数据表存入 Hbase 数据库，并设计数据库的行键和列

III. 设计好的数据库实现所需求的要求

IV. 上机课的 Hbase 基本操作作业过程在附录中

2. 实验过程

N1 和 D1 的自然连接操作是等值连接，需要存在一个属性相同的列，观察 D1 表只有项目编号和项目名称，而在 N1 表中只有项目编号的信息，但是这个信息却不在一个单独的字段中，而是和副标题一起构成了一个字段，所以应该清洗一下 N1 中的数据，为了后续处理的方便，在第一步即清洗 N1 中的数据，将每一个带有信息的数据都处理成单独的字段。

Hadoop 实现 natural join 的思路：

map 阶段：给 D1 和 N1 加上标记，其实就是多输出一个字段，给 N1 加标记 0，给 D1 加标记 1，其中在此阶段将每一个带有信息的数据都处理成单独的字段。

partition 阶段：根据项目编号为第一主键，标记 label 为第二主键进行排序和分区，reduce 阶段：已经按照第一主键、第二主键排好了序，将相邻相同 key 数据合并输出。

使用 python 以及 hadoop streaming 完成。

mapper.py 和 reducer.py 如下：

Open ▾ mapper.py ~/ Save

create.txt × mapper.py ×

```
#!/usr/bin/python
import sys
import os
#distinguish two input files
filepath = os.environ["map_input_file"]
filename = os.path.split(filepath)[-1]
#lines = read.input(sys.stdin)
for line in sys.stdin:
    if filename == 'D1.txt':
        for ele in line.strip().split(' '):
            if '.' in ele:
                name = ele.split('.')[0]
                file_type = ele.split('.')[1]
            elif '/' in ele:
                year = ele.split('/')[0]
                month = ele.split('/')[1]
                day = ele.split('/')[2]
            elif ':' in ele:
                time = ele
            elif ',' in ele:
                size = ele.replace(',','')
            elif '-' in ele:
                number = ele.split('-')[0]
                sub_number = ele.split('-')[1]

print('\t'.join((number,'1',sub_number,name,file_type,year,month,day,time,si
    elif filename == 'N1.txt':
        line = line.strip()
        number = line.split(' ')[0]
        project = line.split(' ')[1]
        print('\t'.join((number,'0',project)))
```

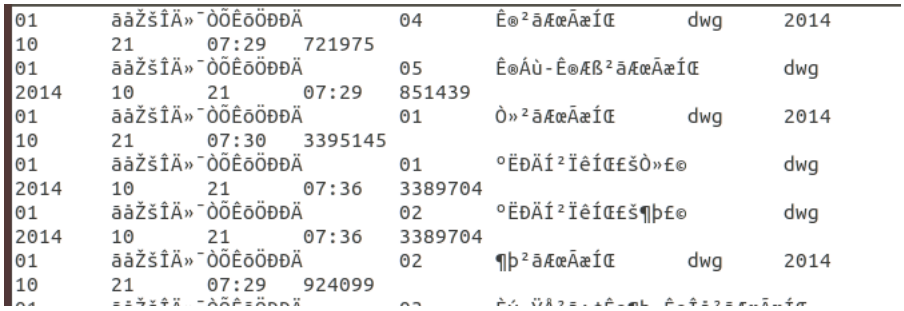
```
#!/usr/bin/python
import sys

last_number = ''

for line in sys.stdin:
    fields = line.split('\t')
    number = fields[0]
    if number != last_number:
        if fields[1] == '0':
            project = fields[2].strip()
            last_number = number
    else:
        sub_number = fields[2]
        name = fields[3]
        file_type = fields[4]
        year = fields[5]
        month = fields[6]
        day = fields[7]
        time = fields[8]
        size = fields[9].strip()

print('\t'.join((number,project,sub_number,name,file_type,year,month,day,tim
```

使用 hadoop streaming

```
  208 ./hadoop jar /usr/local/Hadoop/share/hadoop/tools/lib/hadoop-streami
ng-2.7.7.jar -D num.key.fields.for.partition=1 -D stream.num.map.output.key
.fields=2 -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitione
r -mapper 'python mapper.py' -file /home/hadoop/mapper.py -reducer 'python
reducer.py' -file /home/hadoop/reducer.py -input input1/* -output output9
```

Stream.num.map.output.key.fields=2 - partitioner 设置 map 的输出以前两个字段作为排序依据，这个配置很重要，否则 reduce 的输入格式不正确。

在 hdfs 下查看输出文件，发现存在编码问题，此部分还未处理，将在后面部分处理，得到如下结果（部分）



此部分需要自己设计数据库，为了满足要求“按文件名和项目编号可快速查询到文件信息”，将文件名和项目编号组合成行键，同时这样的设计也可以避免文件名重复而造成的行键重复。在 hbase 中使用简单的 scan 或者 get 操作即可获得信息。为了满足“按项目编号分组输出”“按图纸年份分组输出”，可知需要将项目编号与年份设计为列，但编号是属于项目信息的一部分，年份是处于时间信息的一部分，故我将数据库设计为：行键：编号-名称　列族：项目（名称、编号、副编），列族：文件信息（类型、大小）列族：时间信息（年、月、日、时间）

接着需要对 hbase 进行 shell 操作生成这个数据库，但是数据量庞大，不应该手动输入，则需要利用脚本进行批量命令行操作，脚本文件通过 python 程序生成。

Python 文件如下：

```
f = open("create_hbase.txt",'w')
f1 = open("part-00000",'r')
create = 'create \'Project-file\',\'Project\',\'File\',\'Time\''
f.write(create)
f.write("\n")
for line in f1:
    put = "put \'Project-file\' "
    fields = line.split("\t")
    put = put + ",\'" + fields[0] + "-" + fields[3] + "\'"
    command1 = put + ",\'Project:name\'" + ",\'" + fields[1] + "\'"
    f.write(command1)
    f.write("\n")
    command2 = put + ",\'Project:number\'" + ",\'" + fields[0] + "\'"
    f.write(command2)
    f.write("\n")
    command3 = put + ",\'Project:subnumber\'" + ",\'" + fields[2] + "\'"
    f.write(command3)
    f.write("\n")
    command4 = put + ",\'File:type\'" + ",\'" + fields[4] + "\'"
    f.write(command4)
    f.write("\n")
    command5 = put + ",\'File:size\'" + ",\'" + fields[9].strip() + "\'"
    f.write(command5)
    f.write("\n")
    command6 = put + ",\'Time:year\'" + ",\'" + fields[5] + "\'"
    f.write(command6)
    f.write("\n")
    command7 = put + ",\'Time:month\'" + ",\'" + fields[6] + "\'"
    f.write(command7)
    f.write("\n")
    command8 = put + ",\'Time:day\'" + ",\'" + fields[7] + "\'"
    f.write(command8)
    f.write("\n")
    command9 = put + ",\'Time:time\'" + ",\'" + fields[8] + "\'"
    f.write(command9)
    f.write("\n")
f.write("get \'Project-file\',\'01-十六-十七层平面图\',{COLUMN=>\'Time:time\'}")
f.write("\n")
f.write("exit")
f.close()
```

此时进行编码处理

```
231  iconv -f gbk -t utf8 create_hbase.txt > create.txt
```

脚本部分如下

```
create 'Project-file','Project','File','Time'
put 'Project-file' ,'01-十层平面图','Project:name','沐川文化艺术中心'
put 'Project-file' ,'01-十层平面图','Project:number','01'
put 'Project-file' ,'01-十层平面图','Project:subnumber','04'
put 'Project-file' ,'01-十层平面图','File:type','dwg'
put 'Project-file' ,'01-十层平面图','File:size','721975'
put 'Project-file' ,'01-十层平面图','Time:year','2014'
put 'Project-file' ,'01-十层平面图','Time:month','10'
put 'Project-file' ,'01-十层平面图','Time:day','21'
put 'Project-file' ,'01-十层平面图','Time:time','07:29'
put 'Project-file' ,'01-十六-十七层平面图','Project:name','沐川文化艺术
put 'Project-file' ,'01-十六-十七层平面图','Project:number','01'
put 'Project-file' ,'01-十六-十七层平面图','Project:subnumber','05'
```

然后启动 hbase 并在 shell 中运行脚本

```
./hbase shell /home/hadoop/create.txt
```

检查表的创建

```
hbase(main):001:0> list
TABLE
Project-file
result
2 row(s)
Took 0.8159 seconds
=> ["Project-file", "result"]
```

查询文件信息只需要构建行键在 hbase 中进行查询即可，类似这样的语句

get 'Project-file','01-十六-十七层平面图',{COLUMN=>'Time:time'}

主要考虑分组输出的问题，很明显这样的问题是很适合使用 mapreduce 形式来处理的，此时需要用 mapreduce 直接操作 hbase，在网上查找资料之后发现 hbase 有继承 JAVA 中的 mapper 和 reducer 类可以进行操作，参考网络上的代码写 java 程序进行 mapreduce。

代码如下：

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.IdentityTableReducer;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.mapreduce.TableReducer;


public class GroupBy {
    public static class ProjectNumberMapper extends TableMapper<Text, Text> {
        public static final byte[] CF = "Project".getBytes();
        public static final byte[] COL1 = "number".getBytes();
        private Text text = new Text();
        private Text text2 = new Text();
        public void map(ImmutableBytesWritable row, Result value, Context context) throws InterruptedException, IOException {
            String val = new String(value.getValue(CF, COL1));
            val = val;
            text.set(val);
            String val2 = new String(value.getRow());
            text2.set(val2);
            context.write(text, text2);
        }

    }
    public static class YearMapper extends TableMapper<Text, Text> {
        public static final byte[] CF = "Time".getBytes();
        public static final byte[] COL1 = "year".getBytes();
        private Text text = new Text();
        private Text text2 = new Text();
        public void map(ImmutableBytesWritable row, Result value, Context context) throws InterruptedException, IOException {
            String val = new String(value.getValue(CF, COL1));
            val = val;
            text.set(val);
            String val2 = new String(value.getRow());
            text2.set(val2);
            context.write(text, text2);
        }

    }
```

```
    public static class MyTableReducer extends TableReducer<Text, Text, ImmutableBytesWritable> {
        public static final byte[] CF = "cf".getBytes();
        public static final byte[] NUM = "num".getBytes();

        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            String concated_values = new String();
            for (Text val : values) {
                concated_values += val.toString()+"\n";
            }
            Put put = new Put(Bytes.toBytes(key.toString()));
            put.addColumn(CF, NUM, Bytes.toBytes(concated_values));

            context.write(null, put);

        }
    }
    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
        Class<? extends TableMapper> mapperClass;
        System.out.println(args[0]);
        if (args[0].equals("number")){
            mapperClass = ProjectNumberMapper.class;
        }
        else if (args[0].equals("year")){
            mapperClass = YearMapper.class;
        }
        else{
            throw new IOException("no args");
        }

        Configuration config = HBaseConfiguration.create();

        Connection connection = ConnectionFactory.createConnection(config);
        Admin admin = connection.getAdmin();
        admin.disableTable(TableName.valueOf("result"));
        admin.truncateTable(TableName.valueOf("result"), true);
        admin.close();

        Job job = Job.getInstance(config,"ExampleReadWrite");
        job.setJarByClass(GroupBy.class);    // class that contains mapper

        Scan scan = new Scan();
        scan.setCaching(500);
        scan.setCacheBlocks(false);


        TableMapReduceUtil.initTableMapperJob(
                "Project-file",      // input table
                scan,              // Scan instance to control CF and attribute selection
                mapperClass,   // mapper class
                Text.class,             // mapper output key
                Text.class,             // mapper output value
                job);


        TableMapReduceUtil.initTableReducerJob(
                "result",      // output table
                //IdentityTableReducer.class,             // reducer class
                MyTableReducer.class,    // reducer class
                job);
        job.setNumReduceTasks(1);

        boolean b = job.waitForCompletion(true);
        if (!b) {
            throw new IOException("error with job!");
        }

        Table table = connection.getTable(TableName.valueOf("result"));
        ResultScanner scanner = table.getScanner(scan);
        for (Result res:scanner){
            System.out.println(new String(res.getRow(), StandardCharsets.UTF_8));
            System.out.println(new String(res.getValue(MyTableReducer.CF, MyTableReducer.ROW)));
        }
        scanner.close();
    }
}
```

在 Eclipse 中下载依赖包并将 java 打包成可执行的 jar 文件。

在 Hadoop 下执行 jar 文件进行分组输出

按照年份

```
hadoop@ubuntu:/usr/local/Hadoop/bin$ ./hadoop jar /home/hadoop/HW5.jar year
```

结果（部分）

```
2014
02-二十层平面图
02-十九层平面图
01-一层平面图
10-核心筒详图（六）
04-避难层防火分区图
10-核心筒详图（五）
10-核心筒详图（三）
01-核心筒详图（二）
02-三十层平面图
10-核心筒详图（四）
08-核心筒详图（八）
04-标准层防火分区图
04-二层防火分区图
04-三十层防火分区图
04-一层防火分区图
01-核心筒详图（一）
03-二十二-二十九层平面图
03-A-N立面图
03-21-15立面图
02-机房层、屋顶平面图
03-15-21立面图
02-屋面层平面图
08-核心筒详图（七）
08-典型幕墙详图
01-三-九层、十二-十五层平面图
01-二层平面图
02-十八层平面图
17-N-A立面图
17-1-1剖面
01-十六-十七层平面图
01-十层平面图
02-二十一层平面图

2015
03-2-2剖面图
10-3号楼卫生间详图_t3

2016
18-楼梯详图5_t3
03-1-1剖面图
```

按照项目编号：

```
hadoop@ubuntu:/usr/local/Hadoop/bin$ ./hadoop jar /home/hadoop/HW5.jar numb
er
```

结果（部分）：

```
01
01-十六-十七层平面图
01-一层平面图
01-核心筒详图（二）
01-核心筒详图（三）
01-十层平面图
01-三-九层、十二-十五层平面图
01-二层平面图

02
02-三十层平面图
02-东立面图
02-一层平面图
02-二十一层平面图
02-二十层平面图
02-二层平面图
02-二层平面图-3号楼二层平面图
02-北立面图
02-十九层平面图
02-十八层平面图
02-屋面层平面图
02-屋顶层平面图
02-屋顶层平面图-3号楼屋顶平面图
02-机房层、屋顶平面图

03
03-2-2剖面图
03-1-1剖面图
03-15-21立面图
03-21-15立面图
03-A-N立面图
03-二十二-二十九层平面图
03-立剖面_t3-160919

04
04-一层防火分区图
04-三十层防火分区图
04-二层防火分区图
04-标准层防火分区图
```

3. 实验问题

此次作业一开始在 Hbase 单机模式下运行，但是单机模式在创建表时总是出错，改为伪分布式后即可成立，还未找到原因。

在 hbase 上执行 mapreduce 时，代码参考网络，自己完全理解代码，更改代码适应于自己创建的数据库进行分组输出。分组输出后按照编号和文件名可在 hbase 下得到信息。

4. 参考资料

https://blog.csdn.net/wild46cat/article/details/53256230

https://blog.csdn.net/yimingsilence/article/details/70242604

https://blog.csdn.net/yang63515074/article/details/80451420

https://blog.csdn.net/Sw_et/article/details/77677943

附录

创建表并插入数据：

```
hbase(main):012:0> create 'Student','S_Name','S_Sex','S_Age'
Created table Student
Took 0.7555 seconds
=> Hbase::Table - Student
hbase(main):013:0> put 'Student','2515001','S_Name','Zhangsan'
Took 0.0341 seconds
hbase(main):014:0> put 'Student','2515001','S_Sex','male'
Took 0.0109 seconds
hbase(main):015:0> put 'Student','2515001','S_Age','23'
Took 0.0069 seconds
hbase(main):016:0> put 'Student','2515002','S_Name','22'
Took 0.0056 seconds
hbase(main):017:0> put 'Student','2515002','S_Sex','female'
Took 0.0159 seconds
hbase(main):018:0> put 'Student','2515002','S_Name','Mary'
Took 0.0042 seconds
hbase(main):019:0> put 'Student','2515002','S_Age','22'
Took 0.0065 seconds
hbase(main):020:0> put 'Student','2515003','S_Name','Lisi'
Took 0.0035 seconds
hbase(main):021:0> put 'Student','2515003','S_Sex','male'
Took 0.0109 seconds
hbase(main):022:0> put 'Student','2515003','S_Age','24'
Took 0.0099 seconds
```

列出 HBase 所有的表的相关信息

```
hbase(main):027:0> list
TABLE
Student
1 row(s)
Took 0.0149 seconds
=> ["Student"]
```

在终端打印学生版的所有记录

```
hbase(main):026:0> scan 'Student'
ROW                   COLUMN+CELL
 2515001              column=S_Age:, timestamp=1555337621797, value=23
 2515001              column=S_Name:, timestamp=1555337602112, value=Zhangsan
 2515001              column=S_Sex:, timestamp=1555337613615, value=male
 2515002              column=S_Age:, timestamp=1555337671962, value=22
 2515002              column=S_Name:, timestamp=1555337787633, value=Mary
 2515002              column=S_Sex:, timestamp=1555337647293, value=female
 2515003              column=S_Age:, timestamp=1555337699726, value=24
 2515003              column=S_Name:, timestamp=1555337683750, value=Lisi
 2515003              column=S_Sex:, timestamp=1555337692401, value=male
3 row(s)
```

向学生表添加课程列族（并输入信息）

```
hbase(main):031:0> alter 'Student',NAME => 'course',VERSIONS => 3
Updating all regions with the new schema...
All regions updated.
Done.
Took 1.3288 seconds
hbase(main):032:0> enable 'Student'
Took 0.7779 seconds
hbase(main):033:0> put 'Student','2515002','course','Chinese'
Took 0.0031 seconds
hbase(main):034:0> put 'Student','2515003','course','Math'
Took 0.0024 seconds
hbase(main):035:0> put 'Student','2515001','course','Math'
```

将课程族中的数学更换为物理

```
hbase(main):037:0> put 'Student','2515003','course','Physics'
Took 0.0076 seconds
hbase(main):038:0> put 'Student','2515001','course','Physics'
Took 0.0079 seconds
```

统计表的行数

```
hbase(main):039:0> count 'Student'
3 row(s)
```

删除年龄列

```
hbase(main):040:0> disable 'Student'
Took 0.4463 seconds
hbase(main):041:0> alter 'Student',{NAME => 'S_Age',METHOD => 'delete'}
Updating all regions with the new schema...
All regions updated.
Done.
Took 1.2789 seconds
hbase(main):042:0> enable 'Student'
Took 0.7449 seconds
```

统计表的列数

```
hbase(main):043:0> describe 'Student'
Table Student is ENABLED
Student
COLUMN FAMILIES DESCRIPTION
{NAME => 'S_Name', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSI
ON_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => '
false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', R
EPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false',
 IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPE
N => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'
}
{NAME => 'S_Sex', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSIO
N_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'f
alse', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', RE
PLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false',
IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN
 => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
{NAME => 'course', VERSIONS => '3', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSI
ON_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => '
false', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', R
EPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'false',
 IN_MEMORY => 'false', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPE
N => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'
}
3 row(s)
```