东北大学秦皇岛分校

《医疗数据系统构建》

课程设计报告

基于 Hadoop 生态的网络流量日志分时分析系统

学　　院	数学与统计学院
专　　业	数据科学与大数据技术
班级序号	2002021
学　　号	202015140
姓　　名	周华
指导教师	王子健、张建波
开始日期	2022 年 12 月 21 日
结束日期	2022 年 12 月 28 日

教师评阅意见书

一、态度 1．工作态度（□认真、□较好、□一般、□较差） 2．出勤情况（□无缺勤、□缺勤不超过1/3、□缺勤超过1/3） 3．上交时间（□按时、□迟交1天、□迟交1天以上）
二、格式规范 4．文字部分（□符合规范、□较符合规范、□一般、□不符合规范） 5．图表部分（□符合规范、□较符合规范、□一般、□不符合规范） 6．数学公式（□符合规范、□较符合规范、□一般、□不符合规范） 7．参考文献（□符合规范、□较符合规范、□一般、□不符合规范）
三、报告内容 8．任务量和可行性（□合理、□较为合理、□一般、□不合理） 9．报告结构（□合理、□较合理、□一般、□不合理） 10．文字叙述（□清晰流畅、□较为清晰、□一般、□不清晰） 11．图表准确性（□准确、□较准确、□一般、□不准确）
四、综合能力 12．综合运用知识能力（□很强、□较强、□一般、□较弱） 13．实践与动手能力（□很强、□较强、□一般、□较弱） 14．创新意识（□很强、□较强、□一般、□较弱）
综合评价：□优秀、□良好、□中等、□及格、□不及格

评阅教师签字：　　日期：

1 绪论

1.1 大数据发展概述

随着云计算，物联网技术的兴起，以及芯片技术和移动应用的发展，数据呈指数型式爆炸性增长，无处不在的数据渗透到了生活的方方面面，大数据时代已经来临。

大数据的定义本身就是一个“大数据”，其定义并没有统一的说法，麦肯锡[1]在其报告《Big data: The next frontier for innovation, competition and productivity》定义大数据指的是大小超出常规的数据库工具获取、存储、管理和分析能力的数据集。Viktor Mayer-Schönberger[2]则在其著作中描述了大数据的 4V 特性:Volume、Velocity、Variety、Value。维基百科对大数据的定义则简单明了: 大数据是指利用常用软件工具捕获、管理和处理数据所耗时间超过可容忍时间的数据集[3]。

无论从大数据的定义还是特性来说，在有限的时间下，对海量的数据进行处理，挖掘出最有价值的信息都是一件具有意义的事，然而随着社会的发展，如何从规模越来越大，增长速度越来越快的海量数据中及时挖掘有价值的信息成为挑战。

1.2 hadoop 发展概述

Hadoop 是 Apache 软件基金会下的一个基于 Java 语言开发的分布式计算平台，被公认为行业大数据标准软件。在分布式环境下具有处理海量数据的能力。随着社会的发展，主流厂商为 Hadoop 开发了更多的开发工具和组件，构成了 hadoop 生态系统。

图 1.1 hadoop10 年发展

2004 年，Nutch 模仿谷歌公司分布式文件系统 (GFS)[4] 论文开发出了自己的分布式文件系统（NDFS），后经发展成为 HDFS。2005 年，Nutch 实现谷歌关于 MapReduce 分布式编程[5] 思想的论文，开发出 MapReduce。2006 年，NDFS 和 MapReduce 从 Nutch 中独立出来，成为 Lucene 的子项目，称为 Hadoop[6]。后续在 Hadoop 的基础上开发了 Yarn，Zookeeper，Hbase，Hive，Pig，Sqoop，Flume 等组件。

1.3 研究思路

Hadoop 的部署模式包括本地模式、伪分布式模式、完全分布式模式三种，本文第一步使用虚拟机为 Hadoop 部署伪分布模式，在 Windows 端搭建 Hadoop 的客户端并配置 Windows 与 Linux 的网络连接。

第二步使用 Hadoop 系统对数据进行处理与分析，首先将日志文件上传至 HDFS，并将文件导入到 HBase 中，导入过程中使用 HMapReduce 对数据进行清洗，然后将 Hive 中新建的表和 HBase 的表建立外部表关系，并对数据进行分析，使用 Sqoop 将分析结果导入到 Mysql 中。研究思路如下：

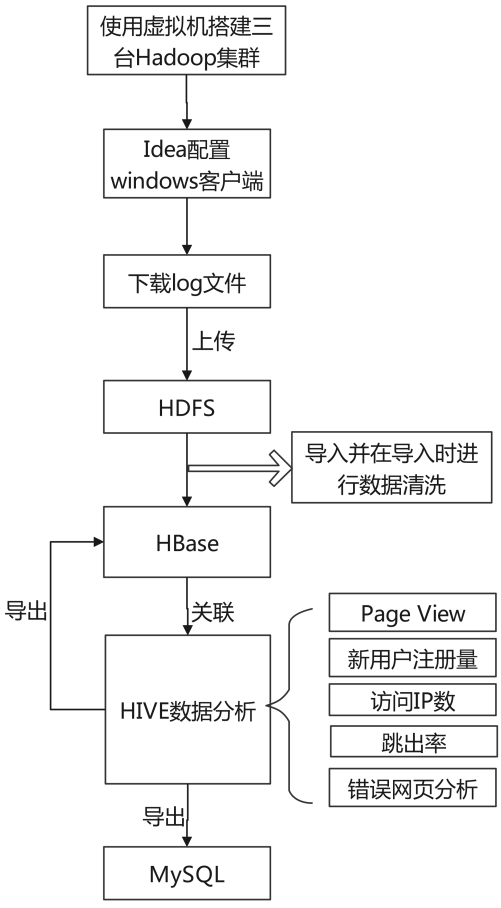

图 1.2　研究思路

2 Hadoop 系统搭建

Hadoop 的部署模式包括本地模式、伪分布式模式、完全分布式模式三种，本文通过在本地搭建虚拟机模拟 Hadoop 集群，从而实现 Hadoop 的伪分布模式部署和远程控制模拟，Hadoop 系统的总体框架图如下：

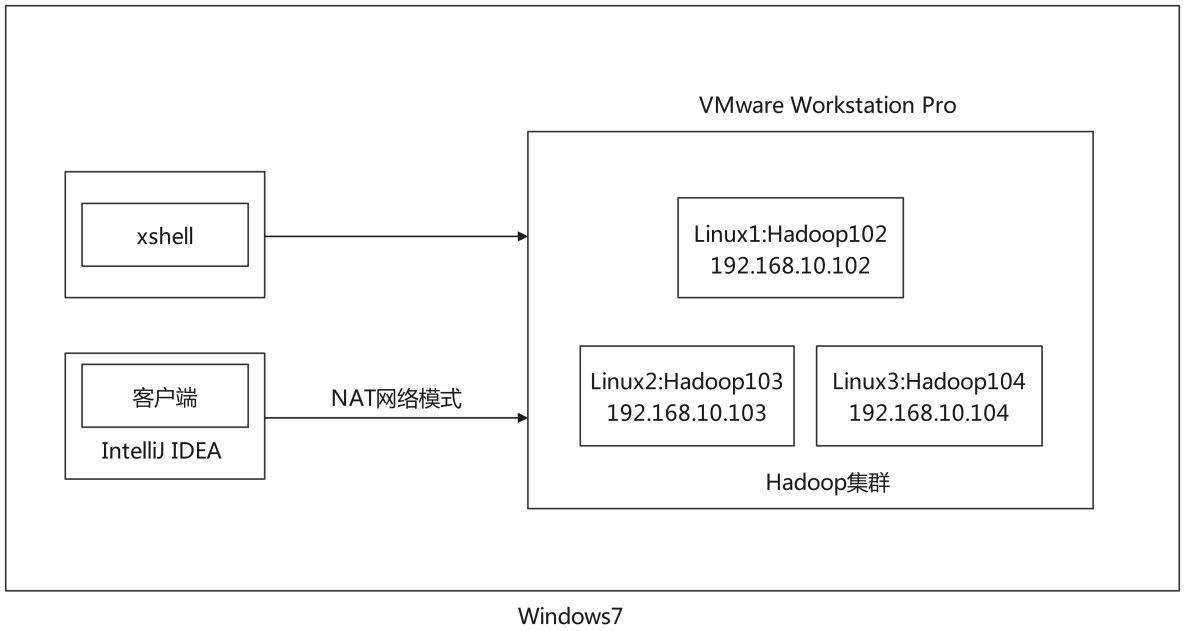

图 2.1 系统整体框架

2.1 Hadoop 集群的搭建

Hadoop 集群的搭建步骤如下：

- 安装 VMware Workstation Pro 虚拟机；
- 以 CentOS-7.5-x86-1804 为镜像安装一台虚拟机；关闭防火墙，创建新用户和密码，卸载系统 JDK；
- 克隆两台虚拟机，设置三台虚拟机的 IP 地址为 192.168.10.102，192.168.10.103，192.168.10.104 并配置 host 文件；
- 设置虚拟机网络为 NAT 模式并设置网关，DNS，并设置远程免密登录；
- 安装 JDK 并进行环境变量设置，分发给其他两台虚拟机；

- 安装 Mysql 并进行环境变量设置，分发给其他两台虚拟机；
- 按顺序安装 Hadoop，Yarn，Zookeeper，Hbase，Hive，Sqoop 并修改相应配置文件。

2.2 Windows 系统客户端构建

Windows 客户端的搭建步骤如下：

- 安装 JDK1.8 并配置环境变量；
- 安装 IntelliJ IDEA 2020.1 x64；
- 配置 Maven 环境；
- 创建 Maven 工程，添加依赖，用 java 语言写出对应的 Hadoop 组件客户端代码，搭建客户端。

3 网络日志大数据分析

3.1 数据集

数据来源于某个网站的后台访问日志，包括两个日志文件：

- 2013_05_30.*log*：2013 年 5 月 30 日的访问日志, 约记录着 50W 条的访问日志;
- 2013_05_31.*log*：2013 年 5 月 31 日的访问日志, 约记录着 140W 条的访问日志。

文件中每一行表示访客访问网站时产生的记录, 从左往右分别为:

- 访客的 ip 地址；
- 访问时间；
- 请求方法；
- 访问的网址路径；
- 网络协议；
- 状态码；
- 传输流量。

3.2 上传日志文件至 HDFS

首先将下载好的日志文件上传至 HDFS，可以通过网页客户端，JavaAPI 客户端，Ftp+Shell 命令三种方式上传文件。

3.2.1 网页客户端上传

通过网页 $http://hadoop102:9870$ 访问网页客户端，在客户端创建 $/lessons/practice/zhouhua$ 和 $/lessons/practice/zhouhua\_202015140/origin\_log$ 文件夹，选择日志文件上传到 $origin\_log$ 文件夹中。

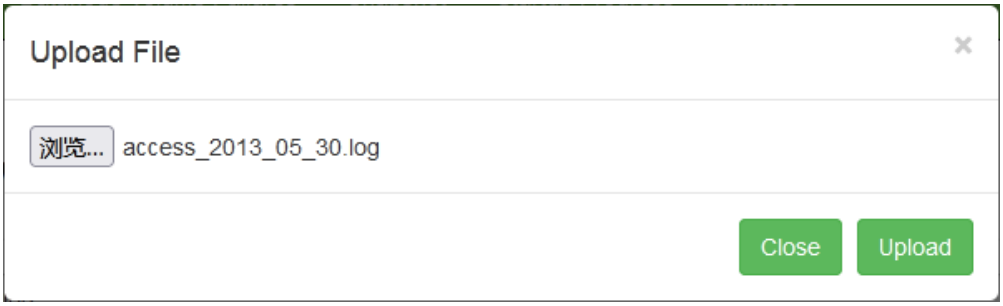

图 3.1 网页客户端上传

3.2.2 ftp+shell 上传

首先使用 Ftp 连接到 $Hadoop102$，将文件发送到 $/opt/$，然后使用 $shell$ 命令将文件上传至 $HDFS$。

```
[atguigu@hadoop102 opt]$ cd /opt/
[atguigu@hadoop102 opt]$ hdfs dfs -cp access_2013_05_30.log /lessons/practice/zhouhua_202015140/origin_log/access_2013_05_30.log
```

图 3.2 ftp+shell 上传

3.2.3 JavaAPI 客户端

启动 Hadoop 集群，在 Idea 中编写上传代码进行上传，关键代码如下，完整代码请参照附录 A。

```java
public static void main(String[] args) throws IOException, URISyntaxException, InterruptedException {
    // 开启HDFS链接
    setUp();
    // 1. 创建工作目录/lessons/practice/周华202015140
    String dirName = "/lessons/practice/周华202015140";
    mkdir( dirName: dirName+"/origin_log");
    mkdir( dirName: dirName+"/cleaned_log");

    // 2. 上传文件，将test.txt文件上传到/lessons/lesson2/zhl/javaApi，调用`fs.copyFromLocalFile()`
    String localPath = "F:/我的个人文件/我的文档/工作文件夹/大数据批处理技术/课程设计/Code/";
    String destPath = "/lessons/practice/周华202015140/origin_log/";
    put( localPath: localPath+"access_2013_05_30.log",  destPath: destPath+"access_2013_05_30.log");
    put( localPath: localPath+"access_2013_05_31.log",  destPath: destPath+"access_2013_05_31.log");
    // 关闭HDFS链接
    tearDown();
}
```

图 3.3　JavaAPI 客户端上传

3.3　将文件导入 HBase

3.3.1　创建数据表

首先在 HBase 中创建两张数据表 *log_zhouhua*_202015140_31 和 *log_zhouhua*_202015140_30，以便导入数据，使用 JavaAPI 进行创建，部分创建代码如下，完整代码参照附录 B。

```java
public static void main(String[] args) throws IOException {
    // 1. 连接HBase数据库
    connectHBase();

    // 2. 创建数据表
    String tableName1 = "log_zhouhua_202015140_31";
    List<String> columnFamilies1 = Arrays.asList("cf1");
    createTable(tableName1, columnFamilies1);

    // 3. 创建数据表
    String tableName2 = "log_zhouhua_202015140_30";
    List<String> columnFamilies2 = Arrays.asList("cf1");
    createTable(tableName2, columnFamilies2);

    // 4. 关闭HBase数据库
    closeHBase();
}
```

图 3.4　创建数据表

3.3.2　数据清理

使用 JavaAPI 将两个日志文件导入到两张表中，并在导入过程中对数据进行清理，Map 代码如下，完整代码参照附录 C。

```
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    // 1. 将每一行读取的数据转换成字符串,将一行数据根据","（英文逗号）分割成字符串数组
    String lineValue = value.toString();
    String[] values = lineValue.split( regex: " ");

    if (values.length == 10) {
        String IP = values[0];String hours = values[3].substring(13, 15);String address = values[6];String result = values[8];

        ImmutableBytesWritable rowKeyWritable = new ImmutableBytesWritable(Bytes.toBytes(key.toString()));

        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn("cf1".getBytes(), "IP".getBytes(), IP.getBytes());
        put.addColumn("cf1".getBytes(), "hours".getBytes(), hours.getBytes());
        put.addColumn("cf1".getBytes(), "address".getBytes(), address.getBytes());
        put.addColumn("cf1".getBytes(), "result".getBytes(), result.getBytes());

        context.write(rowKeyWritable, put); }
    else{
        System.out.println(lineValue);
        String fileName = "D:\\access_2013_05_31_filter.log";
        Path path = Paths.get(fileName);
        try (BufferedWriter writer = Files.newBufferedWriter(path, StandardCharsets.UTF_8, StandardOpenOption.APPEND)){
            writer.write(lineValue);writer.write( str: "\r\n");
        } } } }
```

图 3.5　数据清理

其中，数据清理的步骤如下：

- 清理不规范的数据，即长度不为 10 的数据行；
- 删除传输流量；
- 删除请求方法；
- 删除网络协议；
- 时间只保存小时.

其中，被筛选掉的部分数据如下：

```
173.192.133.53 - - [30/May/2013:18:01:04 +0800] "Connection: close" 400 226
119.147.146.118 - - [30/May/2013:19:59:42 +0800] "GET /static/js/qshare.js\" type=\"text/javascript HTTP/1.1" 403 227
222.128.187.245 - - [30/May/2013:20:43:05 +0800] "2_sendmail=1" 200 15041
58.48.195.42 - - [31/May/2013:08:28:58 +0800] "au, tkdfknnp, xnhftu, zajzauep, vckgxvh, nptxxnhf, hvzae, vgaazja ,hutf" 400 226
222.192.185.133 - - [31/May/2013:08:29:49 +0800] "GET /api.php?mod=js&bid=94  HTTP/1.1" 200 275
60.6.27.223 - - [31/May/2013:12:58:37 +0800] "\x0e\x88\xba\xcb\r\xf8\x05\xb0\x068" 400 226
27.43.27.5 - - [31/May/2013:13:00:42 +0800] "jzauep, vckgxvh, nptxxnhf, hvzae, azja ,hutf" 400 226
42.62.37.26 - - [31/May/2013:13:32:23 +0800] "T 5.1; zh-CN; rv:1.9.1.2) Firefox/3.5.2" 400 226
101.247.48.227 - - [31/May/2013:18:32:50 +0800] "usbcnkalpyininleihghbenswrnzedtjuzeebjlnnrzqzstawvphpivcjivsagsenqwagvqlqqmccmkadc" 200 15041
222.192.185.133 - - [31/May/2013:20:25:27 +0800] "GET /api.php?mod=js&bid=67  HTTP/1.1" 200 2295
119.147.146.118 - - [31/May/2013:20:41:10 +0800] "GET /static/js/qshare.js\" type=\"text/javascript HTTP/1.1" 403 227
180.173.152.175 - - [31/May/2013:21:52:49 +0800] " mxknnp, trkaxm, rnppt, nhftuhv, dtplvfo, jktbntp, ptxxnhf, hvzajzau, tkdfknnp, xnhftu, zajzauep, vckgxvhazja ,hutf" 400 226
```

图 3.6　被筛选掉的部分数据

可以看到存在少部分的日志信息不规范，包括请求方法缺失，网址缺失，网页不正确，也包括网址中含有分隔符空格符号导致分隔不准确而被筛选掉的，30 日文件筛选掉 8 条记录，31 日文件筛选掉 26 条记录，共筛选 34 条记录，认为对总体没有影响。

3.4 使用 Hive 对数据进行分析

3.4.1 建立外部表并关联 HBase 表

启动 hive，建立外部表并关联 HBase 中的两张表，代码如下：

```
CREATE EXTERNAL TABLE log_zhouhua_202015140_30(id int,ip STRING,hours STRING,address STRING,result STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES("hbase.columns.mapping"=":key,cf1:IP,cf1:hours,cf1:address,cf1:result")
TBLPROPERTIES("hbase.table.name"="log_zhouhua_202015140_30");
```

图 3.7　Hive 关联 HBase

其中 id 为数据清理时生成的每行唯一的 row_key。

3.4.2 Page View

使用 Hive 对 30 日、31 日、两日的网络地址进行分组统计，并将结果保存到 $PV1.log$、$PV2.log$、$PVS.log$，其中，30 日，31 日访问页面前 10 如下：

	5_30_address	5_30_counts	5_31_address	5_31_counts
0	/api.php?mod=js&bid=94	5330	/api.php?mod=js&bid=94	17121
1	/static/js/common.js?y7a	3886	/misc.php?mod=patch&action=ipnotice&inajax=1&a...	11608
2	/static/image/common/logo.png	3362	/api.php?mod=js&bid=65	11166
3	/static/image/common/security.png	3321	/member.php?mod=logging&action=login	10170
4	/api.php?mod=js&bid=65	3293	/forum.php	9380
5	/static/image/common/search.png	3231	/static/js/qshare.js	9155
6	/static/image/common/qmenu.png	3209	/static/image/common/logo.png	7838
7	/static/image/common/pt_item.png	3175	/static/js/common.js?F97	7635
8	/static/image/common/arrwd.gif	3172	/static/image/common/search.png	7581
9	/static/js/forum.js?y7a	3125	/static/image/common/qmenu.png	7547

从表中可以看出，$/api.php?mod=js\&bid=94$ 是访问次数最多的页面，并且访问次数最多的大部分是静态网页。

3.4.3 注册量

通过对 30 日，31 日按照小时分组，按网址级数，并筛选出值为 $/member.php?mod=reg$ 的数据，可以得出两天每小时的注册量和总的注册量。数据的汇总结果如下：

hours	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	total
30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	44	42	46	38	30	46	255
31	36	38	26	18	39	35	20	15	86	106	46	70	64	42	40	18	15	17	8	15	7	4	8	7	780

后续根据绘图工具进行绘图结果如下：从图中可以看出，早晨和中午为注册的高峰

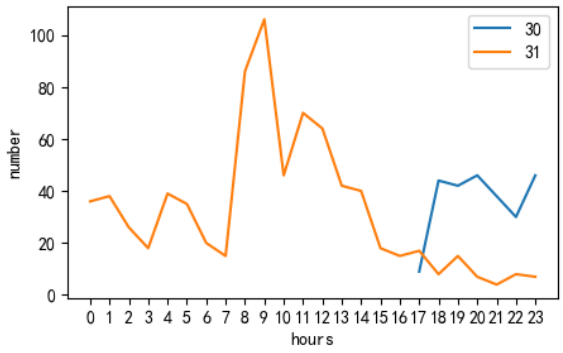

图 3.8　注册量-时间图

期，此时网络可能发生拥堵。

3.4.4　IP 数

通过对 30 日，31 日，及两日的 hour，IP 进行分组并统计数量，可以得出 30 日，31 日，及两日的 IP 数，结果如下：

hours	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	total
30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1241.0	2331.0	2459.0	2573.0	2755.0	2529.0	2315.0	10509.0
31	1838	1351	1213	1063	1020	1050	1199	1364	1994	2486	2572	2463	2449	2311	2472	2513	2545	2643	2132	2140	2069	2293	2279	2236	24779

后续根据绘图工具进行绘图结果如下：从图中可以看出，白天是人们上网的高峰期。

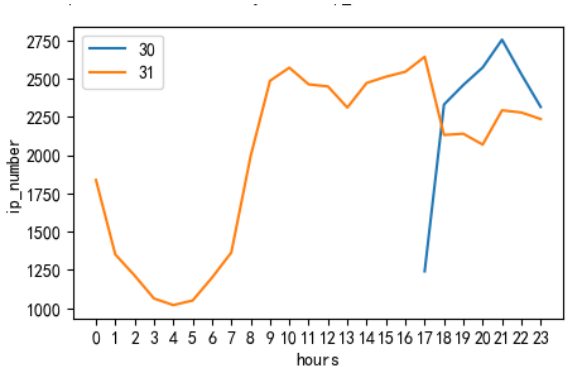

图 3.9　IP 数-时间图

3.4.5 跳出率

通过对 30 日，31 日，及两日的 IP 进行分组并统计数值大于 1 和小于 1 的数据，可以得出 30 日，31 日，及两日的跳出数，结果如下：

	30	31	total
大于 1 次	6792	16408	21200
1 次	3717	8371	10530
跳出率	35.37%	33.78%	33.19%

可以看出有三分之一的用户只浏览了一次网页后就关闭了网页。

3.4.6 404，500 页面

通过对两日数据按网页状态分组，可以统计网页状态为 404，500 的网页的数量，选择出数量最多的 10 个网站，将数据存储到 error.log 中，结果如下：

	404_adress	404_number	500_adress	500_number
0	/static/image/common/bg_pgbtn.png	4838	/thread-11560-1-1.html	16
1	/none	1921	/thread-11560-1-1.html?from=wx	9
2	/images/it315logo.gif	887	/thread-8812-1-4.html	3
3	/static/image/common/scf.cur	770	/thread-12099-1-1.html?from=zygswb	3
4	/static/image/stamp/011.gif	327	/thread-7734-1-4.html	2
5	/xwb/images/bgimg/icon_logo.png	117	/thread-10673-1-3.html	2
6	/archiver/member.php?mod=logging&action=login&...	114	/thread-9633-1-4.html	2
7	/apple-touch-icon-precomposed.png	71	/thread-11265-1-1.html	2
8	/apple-touch-icon.png	67	/home.php?mod=space&uid=17496&mobile=2	2
9	/source/plugin/dsu_kkvip/images/vline.png	64	/thread-11464-1-1.html	2

可以看出，网站的部分资源可能发生了缺失，损坏；当服务器处理多线程网页时，可能会发生服务器错误，因此可以考虑减少线程数量。

3.5 使用 Sqoop 导出数据

使用 Sqoop 可以将数据的结果导出到 MySQL 中，导出的代码如下：

```
sqoop export --connect jdbc:mysql://hadoop102:3306/test \
--username atguigu \
--table log_zhouhua_202015140_31 \
--columns "ip,days,hours,result" \
--update-mode allowinsert \
--update-key id \
```

```
--export-dir /user/hive/warehouse/log_zhouhua_202015140.db/log_31 \
--input-fields-terminated-by "\t"
```

3.6 总结

从整个过程可以看出，Hadoop 通过 MapReduce 这样的分工，让所有的集群分工均匀，从而处理庞大的数据，先将任务细分，再汇合是一种重要的思想。从数据分析结果来看，对于网站而言，有一些页面是访问量巨大的网页，这些页面可以设置成静态网页以提高稳定性；早上和中午时刻是注册量的高峰期，此时注册服务可能发生拥堵；白天是上网人数最活跃的时候；对于出现 404 用户较多的网页，数据可能发生缺失或损坏，需要对网页进行维护。

参考文献

[1] MANYIKA J, CHUI M, BROWN B, et al. Big data: The next frontier for innovation, competition, and productivity[J], 2011.

[2] MAYER-SCHöNBERGER V. Big Data: A Revolution That Will Transform How We Live, Work, and Think[M]. [S.l.] : Mariner Books, 2013.

[3] 孟小峰; 慈祥. 大数据管理: 概念、技术与挑战 [J], 2013 : 146 – 169.

[4] GHEMAWAT S, GOBIOFF H, LEUNG S-T. The Google File System[J/OL]. ACM SIGOPS Operating Systems Review, 2003, 37 : 29 – 43. http://dx.doi.org/10.1145/945445.945450.

[5] DAYALAN M. MapReduce: Simplified Data Processing on Large Cluster[J/OL]. International Journal of Research and Engineering, 2018, 5 : 399 – 403. http://dx.doi.org/10.21276/ijre.2018.5.5.4.

[6] 林子雨. 大数据技术原理与应用 [M]. [S.l.] : 人民邮电出版社, 2017.

附录 A　上传文件代码

HDFSJavaApi.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.IOUtils;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.nio.charset.StandardCharsets;
import java.sql.Timestamp;

public class HDFSJavaApi {

    // 设置HDFS默认访问地址:
    public final static String HDFS_PATH = "hdfs://202.206.19.34:9000";
    // 配置对象声明
    private static Configuration configuration = null;
    // HDFS文件系统对象声明
    private static FileSystem fs = null;

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

        // 开启HDFS链接
        setUp();


        // 1. 创建目录"/lessons/lesson2/zhl/javaApi", 调用'fs.mkdirs()'
//        String dirName = "/lessons/lesson2/zhl/javaApi";
//        mkdir(dirName);

        // 2. 上传文件, 将test.txt文件上传到/lessons/lesson2/zhl/javaApi, 调用'fs.
    copyFromLocalFile()'
//        String localPath = "./test.txt";
//        String destPath = "/lessons/lesson2/zhl/javaApi/test.txt";
//        put(localPath, destPath);
        // 3. 下载文件, 将集群的/lessons/lesson2/zhl/javaApi/test.txt下载到本地,
    调用'copyToLocalFile()'
//        String hdfsPath = "/lessons/lesson2/zhl/javaApi/test.txt";
//        String localPath = "./";
//        get(localPath, hdfsPath);

        // 4. 创建文件并写入数据, 在"/lessons/lesson2/zhl/javaApi/test1.txt", 并写
    入"Hello HDFS test1"
```

```
        //    调用接口'fs.create()'
//        String filePath = "/lessons/lesson2/zhl/javaApi/test1.txt";
//        String content = "Hello HDFS test1111\n";
//        create(filePath, content);


        // 5. 查看文件内容， 查看"/lessons/lesson2/zhl/javaApi/test1.txt", 调用'fs.open()'
//        String filePath = "/lessons/lesson2/zhl/javaApi/test1.txt";
//        read(filePath);


        // 6.对文件追加内容，在"/lessons/lesson2/zhl/javaApi/test1.txt"文件后面追加"Hello Hadoop",
        //   调用'fs.append()'
//        String fileName = "/lessons/lesson2/zhl/javaApi/test1.txt";
//        String appendContent = "Hello Hadoop\n";
//        append(fileName, appendContent);


        // 7.删除文件， 删除"/lessons/lesson2/zhl/javaApi/test1.txt"文件， 调用'fs.deleteOnExit()'
//        String fileName = "/lessons/lesson2/zhl/javaApi/test1.txt";
//        delete(fileName);


        // 8. 遍历文件夹，遍历"/lessons/lesson1/"的文件夹，调用'fs.listStatus()'
//        String filePath = "/lessons/lesson1/";
//        ls(filePath);

        // 9. 查看文件信息， 查看"/lessons/lesson2/zhl/javaApi/test.txt"的元数据信息， 调用'fs.getFileStatus()'
        String filePath = "/lessons/lesson2/zhl/javaApi/test.txt";
        getMetaData(filePath);


        // 关闭HDFS链接
        tearDown();
    }

    // 创建资源
    private static void setUp() throws URISyntaxException, IOException, InterruptedException {
        System.out.println("HDFSApp setUp");
        configuration = new Configuration(); // 创建配置对象
        fs = FileSystem.get(new URI(HDFS_PATH), configuration, "roo"); //文件系统对象
```

```
    }

    // 释放资源
    private static void tearDown() throws IOException {
        // 释放资源
        configuration = null;
        fs.close();
        System.out.println("HDFS app tearDown.");
    }

    // 创建文件夹
    private static void mkdir(String dirName) throws IOException {
        Path path = new Path(dirName);
        boolean isOK = fs.mkdirs(path);
        if (isOK){
            System.out.println(dirName + "创建成功");
        } else {
            System.out.println(dirName + "创建失败");
        }
    }

    // 上传文件
    private static void put(String localPath, String destPath) throws IOException {
        Path local = new Path(localPath); //1.设置本地文件夹
        Path dest = new Path(destPath); //2.设置HDFS文件夹
        fs.copyFromLocalFile(local, dest);
        System.out.println("文件上传成功");
    }

    // 下载文件
    private static void get(String localPath, String hdfsPath) throws IOException {
        fs.copyToLocalFile(new Path(hdfsPath), new Path(localPath)); //设置远程和本地文件
        System.out.println("文件下载成功");
    }

    // 写入数据
    private static void create(String fileName, String content) throws IOException {
        Path path = new Path(fileName); //1. 新建目标文件路径
        FSDataOutputStream outputStream = fs.create(path, (short) 2);//2. 创建文件并获取文件输出流对象，并设置备份数为2
        outputStream.write(content.getBytes(StandardCharsets.UTF_8));//3. 写入数据
        outputStream.close();//4. 关闭文件输出流
        System.out.printf("文件创建和写入完成");
    }

```

```
    // 查看文件
    private static void read(String filePath) throws IOException {
        Path path = new Path(filePath); //1. 设置目标文件路径
        FSDataInputStream inputStream = fs.open(path);//2. 打开文件到字节流中
        IOUtils.copyBytes(inputStream, System.out, 4096, false);//3. 将文件内容拷贝到
        IOUtils.closeStream(inputStream);
    }

    // 追加内容
    private static void append(String fileName, String appendContent) throws IOException {
        Path path = new Path(fileName); //1. 设置目标文件路径
        FSDataOutputStream outputStream = fs.append(path);//2. 创建文件并获取文件输出流对象
        outputStream.write(appendContent.getBytes(StandardCharsets.UTF_8)); //3. 写入数据
        outputStream.close();//4. 关闭文件输出流
        System.out.println("内容追加成功");
    }

    // 删除文件
    private static void delete(String fileName) throws IOException {
        Path path = new Path(fileName); //1. 设置目标文件路径
        boolean isOk = fs.deleteOnExit(path); // 2. 删除目标文件
        if (isOk){ //3. 打印信息
            System.out.println(fileName + "文件删除成功");
        } else {
            System.out.println(fileName + "文件删除失败");
        }
    }

    // 遍历文件夹
    private static void ls(String dirName) throws IOException {
        Path path = new Path(dirName); // 1.设置目标文件夹
        FileStatus[] fileStatuses = fs.listStatus(path); //2. 获取所有文件（夹）的元数据
        for (FileStatus fileStatus : fileStatuses) { //3. 遍历获取文件（夹）的路径
            Path path1 = fileStatus.getPath();
            System.out.println(path1);
        }
    }

    // 获取文件元数据信息
    private static void getMetaData(String dirName) throws IOException {
        Path path = new Path(dirName);
        FileStatus fileStatus = fs.getFileStatus(path);
```

```
        if (fileStatus.isDirectory()){  // 根据文件类型输出
            System.out.println("这是一个文件夹");
        } else {
            System.out.println("这是一个文件");
        }
        // 输出文件元数据信息
        System.out.println("文件路径: " + fileStatus.getPath());
        System.out.println("文件修改时间: " + new Timestamp(fileStatus.
    getModificationTime()));
        System.out.println("上次访问时间: " + new Timestamp(fileStatus.
    getAccessTime()));
        System.out.println("文件长度: " + fileStatus.getLen());
        System.out.println("文件备份数: " + fileStatus.getReplication());
        System.out.println("文件块大小: " + fileStatus.getBlockSize());
        System.out.println("文件所有者: " + fileStatus.getOwner());
        System.out.println("文件所在分组: " + fileStatus.getGroup());
        System.out.println("文件的权限: " + fileStatus.getPermission().toString());
    }
}
```

附录 B　建表代码

HBaseJavaApi.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class HBaseJavaApi {

    // 数据库连接配置
    private static Configuration conf = HBaseConfiguration.create();

    // 数据库连接实例
    private static Connection connection = null;

    // 数据库管理对象
    private static Admin admin = null;


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

```

```
        // 1. 连接HBase数据库
        connectHBase();


        // 2. 创建数据表
        String tableName = "students_info";
        List<String> columnFamilies = Arrays.asList("personal_info", "office_info")
    ;
        createTable(tableName, columnFamilies);

        // 3. 在表中插入数据
        List<Map<String, String>> data = new ArrayList<>();
        Map<String, String> item = new HashMap<>();
        item.put("name", "张三");
        item.put("city", "北京");
        item.put("phone", "131********");
        item.put("tel", "010-11111111");
        item.put("address", "atguigu");
        data.add(item);
        insertData(tableName, data);

        // 4. 查询数据
        String rowKey = "row_key1";
        getData(tableName, rowKey);

        // 5. 删除数据
        deleteData(tableName, rowKey);

        // 6. 关闭HBase数据库
        closeHBase();
    }

    private static void deleteData(String tableName, String rowKey) throws
    IOException {
        // 新建一个Table对象，传入指定的table名，Table负责与记录相关的操作，如增删
    改查等
        Table table = connection.getTable(TableName.valueOf(tableName));
        // 创建删除对象Delete，根据rowkey删除一整条数据
        Delete delete = new Delete(rowKey.getBytes());
        table.delete(delete);
        // 释放资源
        table.close();
        System.out.println("数据删除成功");
    }

    private static void getData(String tableName, String rowKey) throws IOException
     {
```

```
66          // 新建一个Table对象，传入指定的table名，Table负责与记录相关的操作，如增删改查等
67          Table table = connection.getTable(TableName.valueOf(tableName));
68          // 创建Get对象，Get对象可以根据rowkey查询
69          Get get = new Get(rowKey.getBytes());
70          // 查询数据，取得结果集，保存在Result中
71          Result result = table.get(get);
72          // 循环输出单元格的数据
73          for (Cell cell : result.rawCells()){
74              // 取得当前单元格所属列族的名称
75              String family = new String(CellUtil.cloneFamily(cell));
76              // 取得当前单元格的列名
77              String qualifier = new String(CellUtil.cloneQualifier(cell));
78              // 取得当前单元格的列值
79              String value = new String(CellUtil.cloneValue(cell));
80              // 输出结果
81              System.out.println("列： " + family + ":" + qualifier + "————值:" + value);
82          }
83          // 释放资源
84          table.close();
85          System.out.println("数据查询成功");
86      }
87
88      private static void insertData(String tableName, List<Map<String, String>> data) throws IOException {
89          System.out.println("开始添加数据");
90          // 获取一个Table对象
91          Table table = connection.getTable(TableName.valueOf(tableName));
92          // 新建Put对象并指定ROWKEY
93          Put put = new Put("row_key1".getBytes(StandardCharsets.UTF_8));
94          // 指定列族"personal_info"、列名"name"、值"张三"
95          for (Map<String, String> item : data) {
96              put.addColumn("personal_info".getBytes(), "name".getBytes(), item.get("name").getBytes());
97              put.addColumn("personal_info".getBytes(), "city".getBytes(), item.get("city").getBytes());
98              put.addColumn("personal_info".getBytes(), "phone".getBytes(), item.get("phone").getBytes());
99
100             put.addColumn("office_info".getBytes(), "phone".getBytes(), item.get("phone").getBytes());
101             put.addColumn("office_info".getBytes(), "address".getBytes(), item.get("address").getBytes());
102         }
103         // Table对象调用put操作
104         table.put(put);
```

```
105
106         // 操作执行完成，关闭table
107         table.close();
108         System.out.println("数据插入完毕");
109     }
110
111     private static void createTable(String table, List<String> columnFamilies)
    throws IOException {
112         // 设置HBase表名
113         TableName tableName = TableName.valueOf(table);
114         // 调用tableExists查看数据表是否存在
115         if (admin.tableExists(tableName)){
116             System.out.println("表已存在， 将删除原表");
117             admin.disableTable(tableName);
118             admin.deleteTable(tableName);
119             System.out.println("已删除原表");
120         }
121         // 新建一个HTableDescriptor表的描述对象，用于添加列族
122         HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
123         // 遍历列族，挨个将列族添加到HTableDescriptor对象中
124         for (String cf: columnFamilies){
125             HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(cf);
126             hTableDescriptor.addFamily(hColumnDescriptor);
127         }
128
129         // 调用createTable创建数据表
130         admin.createTable(hTableDescriptor);
131         System.out.println("数据表创建成功");
132
133
134
135     }
136
137     private static void closeHBase() throws IOException {
138         if (admin != null){
139             admin.close();
140             admin = null;
141         }
142         if (connection != null){
143             connection.close();
144             connection = null;
145         }
146
147     }
148
149     private static void connectHBase() throws IOException {
150         // 设置Zookeeper的通信ip和端口，需要与集群中的Zookeeper对应
```

```
        conf.set("hbase.zookeeper.quorum",
                "172.15.0.2:2181,172.15.0.3:2181,172.15.0.4:2181");

        // 设置连接对象
        connection = ConnectionFactory.createConnection(conf);

        // 获取数据库管理对象
        admin = connection.getAdmin();
        System.out.println("数据库连接成功");
    }
}
```

附录 C　数据清理代码

HBaseMRApi.java，ReadHDFSStudentMapper.java，WriteHBaseStudentReducer.java

```
import Case1.Mapper.ReadHDFSStudentMapper;
import Case1.Reducer.WriteHBaseStudentReducer;
import Case2.MyMapper.MyMapper;
import Case2.MyReducer.MyReducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import java.io.IOException;

public class HBaseMRApi{

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

        // 1. 配置HBase的Zookeeper位置
        Configuration conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum","cluster-master:2181,cluster-slave1:2181,
    cluster-slave2:2181");

        // 2. 配置Job实例
        Job job = Job.getInstance(conf, "hbase_mr_job");
        job.setJarByClass(HBaseMRApi.class);
```

```
29
30          // 3. 设置输入文件路径(如果使用TableMapper, 此处应该是为Job初始化Mapper)
31          if (args[0].equals("1")){
32              // 从HDFS中载入数据
33              Path inputPath = new Path("hdfs://cluster-master:9000/lessons/lesson3/
    zhl/data/students_info.csv");
34              job.setInputFormatClass(TextInputFormat.class);
35              TextInputFormat.addInputPath(job, inputPath);
36              // 4. 设置Mapper类
37              job.setMapperClass(ReadHDFSStudentMapper.class);
38              job.setMapOutputKeyClass(ImmutableBytesWritable.class);
39              job.setMapOutputValueClass(Put.class);
40              TableMapReduceUtil.initTableReducerJob("students_info",
    WriteHBaseStudentReducer.class, job);
41          } else {
42              // 从HBase中载入数据
43              TableMapReduceUtil.initTableMapperJob("students_info", new Scan(),
    MyMapper.class, ImmutableBytesWritable.class, Put.class, job);
44              TableMapReduceUtil.initTableReducerJob("students_info1", MyReducer.
    class, job);
45          }
46
47          // 5. 设置Reducer类
48          job.setOutputKeyClass(Put.class);
49          job.setOutputValueClass(NullWritable.class);
50          job.setNumReduceTasks(1);
51
52          // 6. 等待程序结束
53          System.out.println(job.waitForCompletion(true) ? 0: 1);
54      }
55  }
```

```
1   package Case1.Mapper;
2
3   import org.apache.hadoop.hbase.client.Put;
4   import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
5   import org.apache.hadoop.hbase.util.Bytes;
6   import org.apache.hadoop.io.LongWritable;
7   import org.apache.hadoop.io.Text;
8   import org.apache.hadoop.mapreduce.Mapper;
9
10  import java.io.IOException;
11
12
13  /**
14   * 读取HDFS中的文件students_info.txt的数据
15   */
```

```java
public class ReadHDFSStudentMapper extends Mapper<LongWritable, Text,
    ImmutableBytesWritable, Put> {

    /**
     * 接收HDFS中的数据，并处理成能存入HBase的样式
     * @param key map阶段输入key，默认为每一行下标
     * @param value map阶段输入value，为一行的内容
     * @param context 上下文
     */
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {

        // 1. 将每一行读取的数据转换成字符串
        //    将一行数据根据","（英文逗号）分割成字符串数组
        String lineValue = value.toString();
        String[] values = lineValue.split(",");

        // 2. 依此从字符串数组中获取数据
        String rowKey = values[0];
        String name = values[1];
        String city = values[2];
        String phone = values[3];
        String tel = values[4];
        String address = values[5];
        System.out.println(rowKey + "-" + name + "-" + city + "-" + phone + "-" +
    tel + "-" + address);

        // 3. 构造rowkey，rowkey为ImmutableBytesWritable类型
        ImmutableBytesWritable rowKeyWritable = new ImmutableBytesWritable(
                Bytes.toBytes(rowKey)
        );

        // 4. 构造Put对象，可以添加一行的数据
        Put put = new Put(Bytes.toBytes(rowKey));
        put.addColumn("personal_info".getBytes(), "name".getBytes(), name.getBytes
    ());
        put.addColumn("personal_info".getBytes(), "city".getBytes(), city.getBytes
    ());
        put.addColumn("personal_info".getBytes(), "phone".getBytes(), phone.
    getBytes());
        put.addColumn("office_info".getBytes(), "tel".getBytes(), tel.getBytes());
        put.addColumn("office_info".getBytes(), "address".getBytes(), address.
    getBytes());

        // 5. 将数据推送到reduce阶段
        context.write(rowKeyWritable, put);
```

```
    }
}
```

```
package Case1.Reducer;

import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.io.NullWritable;

import java.io.IOException;


/**
 * 将Put操作应用到数据表上
 */
public class WriteHBaseStudentReducer extends
        TableReducer<ImmutableBytesWritable, Put, NullWritable> {

    /**
     * @param key 输入的rowkey
     * @param values 输入的Put对象，为一个可迭代对象
     * @param context 上下文对象
     * @throws IOException
     * @throws InterruptedException
     */
    @Override
    protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context
     context) throws IOException, InterruptedException {


        for (Put put : values){
            context.write(NullWritable.get(), put);
        }
    }
}
```