华东师范大学计算机科学与技术系上机实践报告

课程名称：大数据系统	**指导教师：姚俊杰**	**实践名称：HBase**
姓名 1：	**姓名 2：**	**姓名 3：**
学号 1：	**学号 2：**	**学号 3：**

一 实验背景

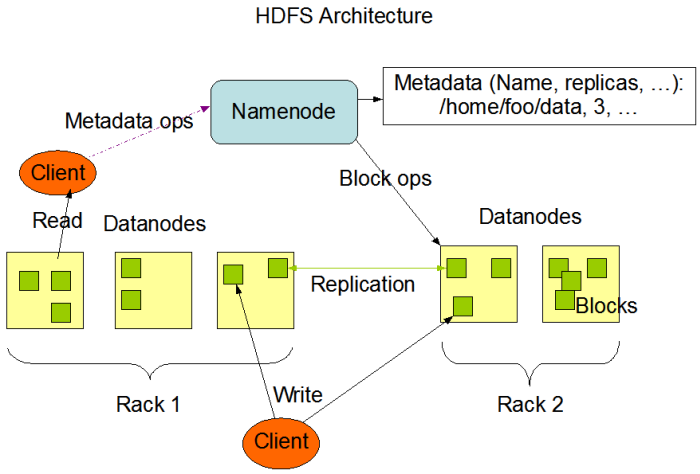
1.1 Hadoop

1.1.1 Hadoop

Apache Hadoop 是一款支持数据密集型分布式应用程序并以 Apache 2.0 许可协议发布的开源软件框架。所有的 Hadoop 模块都有一个基本假设，即硬件故障是常见情况，应该由框架自动处理。Hadoop 框架透明地为应用提供可靠性和数据移动，提供了分布式文件系统用以存储所有计算节点的数据，这为整个集群带来了非常高的带宽。普遍认为整个 Apache Hadoop“平台”包括 Hadoop 内核、MapReduce、HDFS 以及一些相关项目，有 Apache Hive 和 Apache HBase 等。

1.1.2 HDFS

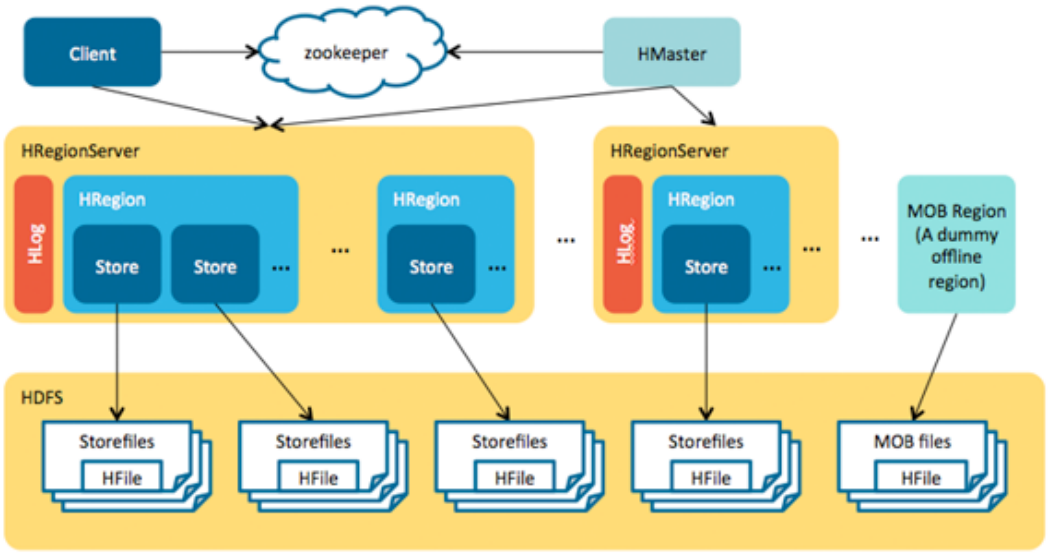
HDFS 具有主/从体系结构。 HDFS 集群由单个管理文件系统名称空间，并控制客户端对文件的访问的主服务器 NameNode 组成。 此外，还有许多用于管理与它们所运行的节点相连的存储结构 DataNode， 它们通常是集群中每个节点一个 HDFS 公开了文件系统名称空间，并允许用户数据存储在文件中。在内部，文件被分成一个或多个块，这些块存储在一组 DataNode 中。NameNode 执行文件系统名称空间操作，还负责确定块到 DataNode 的映射。 DataNode 不仅负责处理来自文件系统客户端的读写请求，还根据 NameNode 的指令执行块创建，删除和复制。



1.2 HBase

1.2.1 HBase

HBase 是一个开源的非关系型分布式数据库(NoSQL)，它参考了谷歌的 BigTable 建模，实现的编程语言为 Java。它是 Apache 软件基金会的 Hadoop 项目的一部分，运行于 HDFS 文件系统之上，为 Hadoop 提供类似于 BigTable 规模的服务。因此，它可以对稀疏文件提供极高的容错率。其结构示意图如下：



1.2.2 HMaster

HBase HMaster 是一个轻量级进程，可将区域分配给 Hadoop 集群中的区域服务器以实现负载平衡。其职能包括：管理和监视 Hadoop 集群，执行管理（用于创建，更新和删除表的接口），控制故障转移，DDL 操作，处理客户的更改架构或更改任何元数据的请求。

1.2.3 Region Server

Region Server 主要是处理来自客户端的数据的读取、写入、更新与删除。Region Server 进程在 Hadoop 群集的每一个节点上运行，具体是在 HDFS 的 DataNode 上。

1.2.4 Zookeeper

HBase 使用 ZooKeeper 作为用于区域分配的分布式协调服务，并通过将它们加载到正在运行的其他区域服务器上来恢复任何区域服务器崩溃。 ZooKeeper 是集中式监视服务器，用于维护配置信息并提供分布式同步。 每当客户希望与区域进行通信时，他们都必须首先联系 Zookeeper。 HMaster 和区域服务器已向 ZooKeeper 服务注册，客户端需要访问 ZooKeeper 仲裁才能与区域服务器和 HMaster 连接。 如果 HBase 群集中的节点发生故障，ZKquoram 将触发错误消息并开始修复故障节点。

二 实验设计

2.0 准备工作

2.0.1 配置用户和用户组

搭建 hadoop 集群环境要求所有主机的用户和用户组要完全一致。对于每台主机，新建 hadoop 用户和 hadoop 用户组，并把 hadoop 用户加入到 hadoop 用户组。

```
sudo adduser hadoop
sudo usermod -a -G hadoop hadoop
```

然后为 hadoop 用户赋予 root 权限，使他可以使用 sudo 命令。

```
sudo vim /etc/sudoers
```

修改/etc/sudoers 文件如下，保存退出，hadoop 用户就拥有了 root 权限。

```
## Allow root to run any commands anywhere
root   ALL=(ALL) ALL
hadoop ALL=(ALL) ALL
```

2.0.2 配置主机名

编辑/etc/hostname，把三台主机名修改为对应名字 master、slave1、slave2，并重启服务器。

编辑/etc/hosts，添加如下映射关系，保存退出。

```
172.17.22.35    master
47.101.137.41   slave1
139.196.183.194 slave2
```

2.0.3 配置免密登录 SSH

为了操作方便，通过设置 SSH 免密码登录，实现三台主机间自由切换。

三台主机发起公钥请求，生成密钥对。

```
ssh-keygen -t rsa
```

master 主机将公钥（~/.ssh/id_rsa.pub 中的内容）复制到文件 authorized_keys 中去。

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

两台从机将公钥复制到主机 master 的~/.ssh/authorized_keys 中去。

```
cat ~/.ssh/id_rsa.pub >> master:~/.ssh/authorized_keys
```

master 主机将~/.ssh/authorized_keys 复制到从机 slave1、slave2 中。

```
scp -r ~/.ssh/authorized_keys slave1:~/.ssh/
scp -r ~/.ssh/authorized_keys slave2:~/.ssh/
```

另外注意，需要设置.ssh 目录的权限为 700，其下文件 authorized_keys 和私钥的权限为 600。否则会因为权限问题导致无法免密码登录。

```
chmod -R 700 .ssh/
sudo chmod 600 .ssh/authorized_keys
```

2.0.4 关闭防火墙

集群需要开放很多端口，因此，为了避免出现端口未开放的问题，需要关闭防火墙。

```
systemctl stop firewalld.service
```

查看防火墙状态，显示 inavtive (dead)，说明防火墙已经关闭。

```
systemctl status firewalld.service
```

2.1 JDK

下载 jdk1.8.0_272。

```
yum install java-1.8.0-openjdk-devel.x86_64
```

在/etc/proflie 中设置环境变量，添加路径如下。

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export
CLASSPATH=.:$JAVA_HOME/lib/rt.jar:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib
```

重新加载配置文件使之生效。

```
source /etc/profile
```

使用 `java -version` 命令检测，显示 java 版本证明环境已配置成功。

```
openjdk version "1.8.0_272"
OpenJDK Runtime Environment (build 1.8.0_272-b10)
OpenJDK 64-Bit Server VM (build 25.272-b10, mixed mode)
```

2.2 zookeeper

2.2.1 安装 zookeeper

在本机上下载 zookeeper 安装包，通过 scp 命令将安装包发送到 master 主机上。

```
scp zookeeper-3.4.10.tar.gz hadoop@47.103.213.21:/home/hadoop/
```

在 master 主机根目录下，将安装包解压至/home/hadoop/下。

```
tar -zxvf zookeeper-3.4.10.tar.gz
```

进入/home/hadoop/中，为了方便日后版本的更新，这里使用软链接的方法。

```
ln -s zookeeper-3.4.10 zookeeper
```

设置环境变量，在~/.bashrc 添加如下内容。

```
#zookeeper
export ZOOKEEPER=/home/hadoop/zookeeper
export PATH=$PATH:$ZOOKEEPER/bin
```

重新加载配置文件使之生效。

```
source ~/.bashrc
```

2.2.2 配置 zookeeper

建立数据和日志文件。

```
mkdir /home/hadoop/zookeeper/data
mkdir /home/hadoop/zookeeper/logs
```

从 conf 目录下拷贝 zoo_sample.cfg 到该目录下并重命名为 zoo.cfg。

```
cp zoo_sample.cfg zoo.cfg
```

修改 zoo.cfg 文件，添加数据和日志路径，以及服务器对应主机和通信端口，其中 2888 端口是 server 和集群中的 leader 交换消息所使用的端口，3888 端口是选举 leader 时所使用的端口。

```
dataDir=/home/hadoop/zookeeper/data
dataLogDir=/home/hadoop/zookeeper/logs

quorumListenOnAllIPs=true
server.0=47.103.213.21:2888:3888
server.1=47.101.137.41:2888:3888
server.2=139.196.183.194:2888:3888
```

在数据目录/home/hadoop/zookeeper/data 下新建 myid 的文件，各个主机对应的内容是不同的，master 的内容是 0，slave1 的内容是 1，slave2 的内容是 2，分别对应 server.x 中的 x。

使用 scp 命令，将配置好的 zookeeper 发送到其他从节点上去。注意修改 myid 文件内容!

```
scp -r /home/hadoop/zookeeper/ slave1:home/hadoop/
scp -r /home/hadoop/zookeeper/ slave2:home/hadoop/
```

在各个节点的/home/hadoop/zookeeper/bin 以下执行命令，启动 zookeeper，并查看状态。

```
zkServer.sh start
zkServer.sh status
```

由于所有节点同时启动，因此选举编号最大的节点，即 server.2 作为 leader。

2.3 hadoop

2.3.1 安装 hadoop

在本机上下载 hadoop 安装包，通过 scp 命令将安装包发送到 master 主机上。

```
scp hadoop-2.7.3.tar.gz hadoop@47.103.213.21:/home/hadoop/
```

在 master 主机根目录下，将安装包解压至/home/hadoop/下。

```
tar -zxvf hadoop-2.7.3.tar.gz
```

进入/home/hadoop/中，为了方便日后版本的更新，这里使用软链接的方法。

```
ln -s hadoop-2.7.3 hadoop
```

设置环境变量，在~/.bashrc 添加如下内容。

```
# hadoop
export HADOOP_HOME=/home/hadoop/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

重新加载配置文件使之生效。

```
source ~/.bashrc
```

使用 hadoop version 命令检测，显示 hadoop 版本证明环境已配置成功。

```
Hadoop 2.7.3
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r
baa91f7c6bc9cb92be5982de4719c1c8af91ccff
Compiled by root on 2016-08-18T01:41Z
Compiled with protoc 2.5.0
From source with checksum 2e4ce5f957ea4db193bce3734ff29ff4
This command was run using
/home/hadoop/hadoop/share/hadoop/common/hadoop-common-2.7.3.jar
```

2.3.2 配置 hadoop

进入 hadoop 的配置目录 home/hadoop/hadoop/etc/hadoop，新建以下几个文件夹。

```
mkdir tmp
mkdir hdfs
mkdir hdfs/name
mkdir hdfs/data
```

修改配置文件 hadoop-env.sh、yarn-env.sh，添加 java 路径。

```
# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.272.b10-
1.el7_9.x86_64/jre
```

修改配置文件 core-site.xml。

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>io.file.buffer.size</name>
    <value>4096</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/hadoop/hadoop/tmp</value>
  </property>
</configuration>
```

修改配置文件 hdfs-site.xml。

```
<configuration>
  <property>
```

```
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/home/hadoop/hadoop/hdfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/home/hadoop/hadoop/hdfs/data</value>
  </property>
  <property>
    <name>dfs.http.address</name>
    <value>master:50070</value>
  </property>
  <property>
    <name>dfs.secondary.http.address</name>
    <value>master:50090</value>
  </property>
  <property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>false</value>
  </property>
</configuration>
```

修改配置文件 yarn-site.xml。

```
<configuration>
   <property>
      <name>yarn.resourcemanager.hostname</name>
      <value>master</value>
   </property>
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
   <property>
      <name>yarn.resourcemanager.address</name>
      <value>master:8032</value>
   </property>
   <property>
      <name>yarn.resourcemanager.scheduler.address</name>
      <value>master:8030</value>
   </property>
   <property>
      <name>yarn.resourcemanager.resource-tracker.address</name>
      <value>master:8031</value>
   </property>
   <property>
```

```
        <name>yarn.resourcemanager.admin.address</name>
        <value>master:8033</value>
    </property>
    <property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>master:8088</value>
    </property>
</configuration>
```

修改配置文件 mapred-site.xml。

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>

  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>master:10020</value>
  </property>

  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>master:19888</value>
  </property>
</configuration>
```

修改配置文件 slaves。

```
Master
slave1
slave2
```

配置文件修改完以后，使用 scp 命令将 hadoop 文件夹发送到其他从节点上去。

```
scp -r /home/hadoop/hadoop/ slave1:home/hadoop/
scp -r /home/hadoop/hadoop/ slave2:home/hadoop/
```

在 master 主机上运行 hadoop。如果是第一次启动 namenode，需要对 namenode 进行格式化。

```
/home/hadoop/hadoop/bin/hdfs namenode -format
```

在/home/hadoop/hadoop/sbin 以下执行命令，启动 hdsf 和 yarn。

```
start hdsf.sh
start yarn.sh
```

2.4 hbase

2.4.1 安装 hbase

在本机上下载 hbase 安装包，通过 scp 命令将安装包发送到 master 主机上。

```
scp hbase-1.2.4-bin.tar.gz hadoop@47.103.213.21:/home/hadoop/
```

在 master 主机根目录下，将安装包解压至/home/hadoop/下。

```
tar -zxvf hbase-1.2.4-bin.tar.gz
```

进入/home/hadoop/中，为了方便日后版本的更新，这里使用软链接的方法。

```
ln -s hbase-1.2.4-bin.tar.gz hbase
```

设置环境变量，在~/.bashrc 添加如下内容。

```
# hbase
export HBASE_HOME=/home/hadoop/hbase
export PATH=$PATH:$HBASE_HOME/bin
```

重新加载配置文件使之生效。

```
source ~/.bashrc
```

使用 hbase version 命令检测，显示 hbase 版本证明环境已配置成功。

```
HBase 1.2.4
Source code repository git://asf-dev/home/busbey/projects/hbase
revision=67592f3d062743907f8c5ae00dbbe1ae4f69e5af
Compiled by busbey on Tue Oct 25 18:10:20 CDT 2016
From source with checksum b45f19b5ac28d9651aa2433a5fa33aa0
```

2.4.2 配置 hbase

在 hbase 文件夹下，新建 zookeeper_data 和 logs 两个文件夹。

```
mkdir zookeeper_data
mkdir logs
```

进入 hbase 配置目录/home/hadoop/hbase/conf，修改配置文件 hbase-env.sh。

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export HBASE_MANAGES_ZK=false

export HBASE_CLASSPATH=/home/hadoop/hadoop-2.7.3/etc/hadoop

export HBASE_OPTS="-XX:+UseConcMarkSweepGC -XX:-AssumeMP"
```

修改配置文件 hbase-site.xml。

```
<configuration>
   <property>
     <name>hbase.tmp.dir</name>
     <value>/home/hadoop/hbase/tmp</value>
   </property>
   <property>
     <name>hbase.zookeeper.property.dataDir</name>
     <value>/home/hadoop/hbase/zookeeper_data</value>
   </property>
   <property>
     <name>hbase.rootdir</name>
     <value>hdfs://master:9000/hbase</value>
```

```
    </property>
    <property>
      <name>hbase.cluster.distributed</name>
      <value>true</value>
    </property>
    <property>
      <name>hbase.zookeeper.quorum</name>
      <value>master,slave1,slave2</value>
    </property>
</configuration>
```

修改配置文件 regionservers。

```
master
slave1
slave2
```

在/home/hadoop/hbase/bin 目录下执行命令，启动 hbase。

```
start-hbase.sh
```

2.5 数据库实现

2.5.1 建立连接

在/home/hadoop/hbase/bin 目录下，启动 thrift。

```
hbase-daemen.sh start thrif
```

使用 happybase 连接数据库，在本机中安装 happybase 库，实现对数据库的操作。

```
def connect(host="47.103.213.21"):
    connection = happybase.Connection(host)
    return connection
```

2.5.2 导入数据

使用 mobike 数据集，数据格式描述如下，其中 bikeID 为单车编号，bikeType 为单车类型（普通、新款），distNum 为单车状态（使用中、可使用、损坏），lng 和 lat 分别为单车位置的经纬坐标。

time	bikeID	bikeType	distID	distNum	distType	lng	lat	dist
2017-04-22T08:30:06	0216149257#	2	216149257	1	0	121.169936	31.396159	230.0
2017-04-22T08:30:06	0216154849#	2	216154849	1	0	121.172004	31.396509	304.0

由于 hbase 属于一种 NoSQL 数据库，每条记录通过一个键进行索引，选取 bikeID 作为 row key。选择字段 lng 和 lat 存入数据库中，并将 time 作为每条记录的时间戳。由此构建的数据库模式如下。

row key	addr	
	lng	lat
bikeID1		
bikeID2		

根据表结构创建 mobike 表，其中 dict(max_versions=10)}表示每个键最多纪录 10 个不同版本的数据。

```
connection = connect()
connection.create_table("mobike", {"addr": dict(max_versions=10)})
```

由于时间戳格式为 1970 年 1 月 1 日 00 时 00 分 00 秒到记录时间的总秒数，因此以数据集中 time 作为时间戳需要先对时间格式进行转换。

```
def timestamp(datetime):
    timeArray = time.strptime(datetime, "%Y-%m-%dT%H:%M:%S")
    timeStamp = int(time.mktime(timeArray))
    return timeStamp
```

在数据集存放的路径中，首先列出所有文件，并一次进行处理。首先对每个文件解压缩得到 csv 表单，并对 csv 表单中的每行记录处理，将 time 转换格式作为时间戳，将 bikeID 作为 row key，并筛选感兴趣的字段 lng 和 lat 发送个服务器。

```
def importData(connection):
    path = "mobike/20170422"
    table = connection.table("mobike")
    gzFiles = os.listdir(path)
    for file in gzFiles:
        if file == ".DS_Store": continue
        csvPath = os.path.join(path, file)
        csvFile = un_gz(csvPath)
        csvFile = open(csvFile, "r")
        lines = csv.reader(csvFile)
        for line in lines:
            table.put(line[1], {"addr:lng":line[6], "addr:lat":line[7]},
                    timestamp=timestamp(line[0]))
            print(line)
        csvFile.close()
```

在 master 主机上，在/home/hadoop/hbase/bin 目录下执行命令 hbase shell，输入 scan ″mobike″命令扫描，得到如下结果。由于数据集过大，上传需要大量时间，只导入一部分数据进行模拟。根据扫描结果，发现共导入 23175 条数据。

```
 0216684697#          column=addr:lat, timestamp=1492821275, value=31.319321
 0216684697#          column=addr:lng, timestamp=1492821275, value=121.616542
 0216684760#          column=addr:lat, timestamp=1492821090, value=31.38199
 0216684760#          column=addr:lng, timestamp=1492821090, value=121.553959
 0216684814#          column=addr:lat, timestamp=1492821235, value=31.335174
 0216684814#          column=addr:lng, timestamp=1492821235, value=121.613932
 0216684817#          column=addr:lat, timestamp=1492821236, value=31.336655
 0216684817#          column=addr:lng, timestamp=1492821236, value=121.601509
 0216684842#          column=addr:lat, timestamp=1492821234, value=31.335209
 0216684842#          column=addr:lng, timestamp=1492821234, value=121.594393
 5516508173#          column=addr:lat, timestamp=1492821212, value=31.34047
 5516508173#          column=addr:lng, timestamp=1492821212, value=121.452176
23175 row(s) in 8.8100 seconds
```

2.5.3 数据操作

编写 lookupData 函数，根据 row_key 查找表中数据。由于查询的键不一定在数据库中，因此增加异常处理，当查找不到时，显示 error row_key。

```
def lookupData(table, row_key):
    try:
        value = table.row(row_key)
        print(value)
        lng = float(value[b'addr:lng'].decode())
        lat = float(value[b'addr:lat'].decode())
        return lng, lat
    except:
        print('error row_key')
        return
```

修改和删除记录，首先调用 lookupData 函数查看原始版本，执行结束后再次调用 lookupData 函数验证操作是否生效。

```
def updateData(table, row_key, new_data):
    lookupData(table, row_key)
    table.put(row=row_key, data=new_data)
    lookupData(table, row_key)

def deleteData(table, row_key):
    lookupData(table, row_key)
    table.delete(row_key)
    lookupData(table, row_key)
```

2.5.4 数据可视化

尝试将服务器中数据库中的信息进行可视化，由于对前后端交互不熟练，异步完成数据可视化：首先通过 Python 获取数据库中数据保存到本机，然后通过 JavaScript 实现前端可视化。loadData 扫面全表，并将记录保存到本地 csv 文件中。由于 hbase 中，每个字段都是 bytes 格式，因此先转换成 str 再转换成 float 形式进行保存。

```
def loadData(table):
    csvFile = open("mobike.csv", "w", newline="")
    writer = csv.writer(csvFile)
    writer.writerow(("bikeID", "lat", "lng"))
    for key, value in table.scan():
        writer.writerow((key.decode(), float(value[b'addr:lat'].decode()),
                                       float(value[b'addr:lng'].decode())))
```

前端调用 Mapbox GL JS 进行可视化。它使用 WebGL 从矢量图块和 Mapbox 样式渲染交互式地图。在 HTML 文件的<head>中导入 JavaScript 和 CSS 文件。

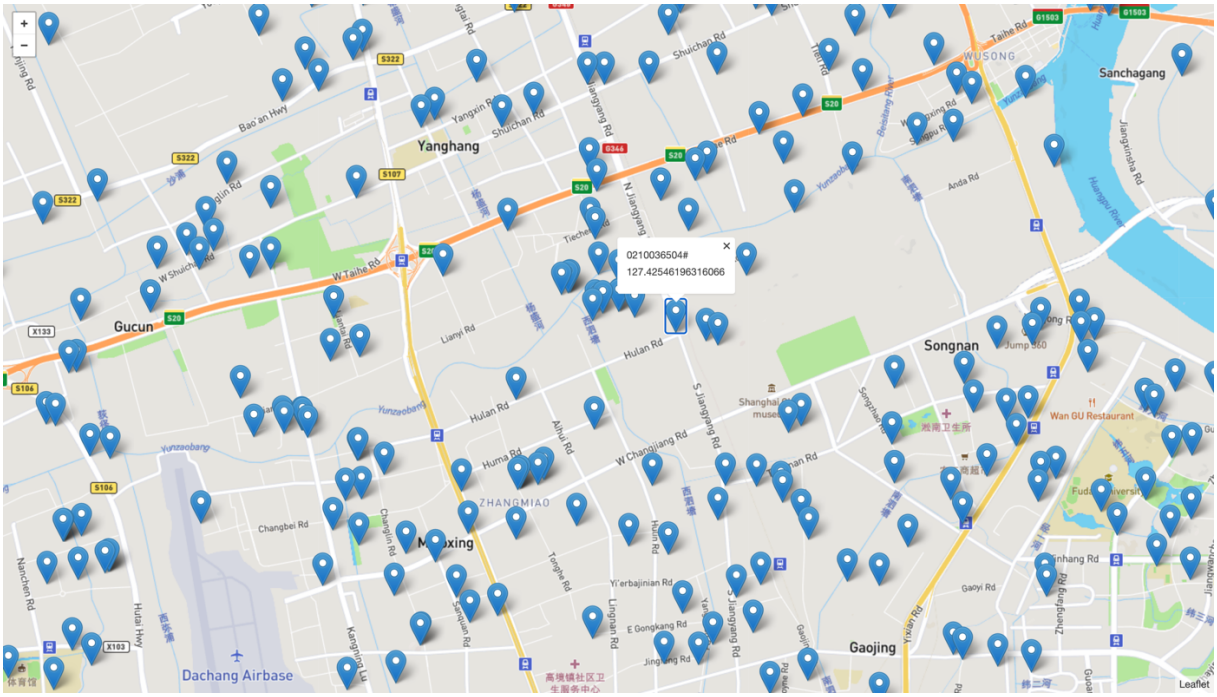
```
<script src='https://api.mapbox.com/mapbox.js/v3.3.1/mapbox.js'></script>
<link href='https://api.mapbox.com/mapbox.js/v3.3.1/mapbox.css'
rel='stylesheet' />
```

在<body>中构建地图，并读取保存在 csv 文件中的数据，将每个位置标注在地图中，每个点可以显示单车编号和距离。

```
var map = L.map('map')
    .addLayer(mapboxTiles)
    .setView([31.4, 121.5], 10);

d3.csv("mobike.csv", function (data) {
    console.log(data);
    for (var i = 0; i < 500; i++) {
        L.marker([data[i]["lat"], data[i]["lng"]]).addTo(map)
          .bindPopup(data[i]["bikeID"] + "<br>" + dist(data[i]))
          .openPopup();
    }
});
```

可视化结果如下。



三 实验调试

在配置环境过程中，我们遇到了很多问题，并且经过了好几个版本的删改，有些问题比较简单，通过网上的一些教程可以比较轻松地解决。但是也遇到了几个查找了对应资料也很难解决，甚至反复卸载重装多次，没有对应解决方法的 BUG，主要列举如下并提供我们的解决方法。

3.1 hadoop 缺少 NameNode

搭建好 zookeeper 和 hadoop 之后，启动 start-all.sh，发现 master 节点所有进程都是正常的，但是没有 NameNode。参考了很多关于 NameNode 无法启动的原因，有很多都是针对 hadoop 中的文

件修改、zookeeper 的不正常关闭以及相关防火墙的设置问题，但是当我们所有状态都是正常，并且两个从节点也是完全正常的时候，我们查找了 dfs 下有两个路径分别对应 NameNode 和 DataNode，但是由于该版本在之前的测试之中运行过，所以对应文件中是有数据的，可能就是因为对应的数据占用了，而导致对应的 NameNode 无法启动。在删除后，再重新运行 start-all.sh 启动成功后，再 jps 测试一下，结果正确。

```
21553 NameNode
21858 SecondaryNameNode
1109 HRegionServer
919 HMaster
21688 DataNode
29178 Jps
2458 QuorumPeerMain
2442 -- process information unavailable
22012 ResourceManager
22126 NodeManager
```

3.2 hbase shell 执行 list status 报错

hbase shell 执行 list status 命令报错：Can't get master address from ZooKeeper; znode data ==null。原因是运行 hbase(zookeeper)的用户无法写入 zookeeper 文件，导致 znode data 为空。hbase-site.xml 文件中的 rootdir 中的 IP 设定很重要，需要设定对应的 IP。添加对应 IP 后解决问题

3.3 从机无法启动 Hregionserver

在启动 hbase 后，从机 slave 无法启动 Hregionserver，网上教程说是三台主机时间不同步，但将三台主机时间同步，同时增大 hbase-site.xml 文件中 maxtimeskew 值，问题依旧没有解决。因此最后并没有实现分布式存储，而是在 master 一个节点中搭建的数据库。