软件综合实习报告

题目：基于 NoSQL 数据库的空间数据存储

院（系）：	计算机学院
专　　业：	计算机科学与技术
姓　　名：	詹才韬
班级学号：	191132
指导教师：	王勇

2016 年 10 月 24 日

目录

1 系统需求分析与总体设计

1.1 需求分析

1.1.1 问题描述

海量空间数据存储要求服务器水平扩展性强，基于 Hadoop 的 NoSQL 数据库具有水平扩展性强的特点，其高效的并行存储机制也为空间检索提供快速响应的能力。

Apache HBase™ is the Hadoop database, a distributed, scalable, big data store. Use Apache HBase™ when you need random, realtime read/write access to your Big Data. This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware. Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS. [1]

Geographic data and information are defined in the ISO/TC 211 series of standards as data and information having an implicit or explicit association with a location relative to the Earth. [2]

1.1.2 方案选择

将空间数据——(x, y)通过希尔伯特曲线降低维数，以此为基础设计 HBase 的行键。设计 HBase 的列簇和列。随机生成二维坐标，插入在 Hadoop 伪分布式环境里面配置好的 HBase 里。设计一个检索查询的方案。设计简洁的 Java 前端图形界面。

1.1.3 需求分析

将空间点数据存储到 NoSQL 数据库——HBase 中，设计空间检索方法，对空间数据进行空间查询。

1.1.4 开发平台

操作系统：	deepin 15.3
文件系统：	HDFS 2.6.4
数据库：	HBase 1.2.2
部署方式：	伪分布式部署
开发语言：	Java
开发工具：	Eclipse

1.2 系统总体设计

1.2.1 前端：Java 客户图形界面

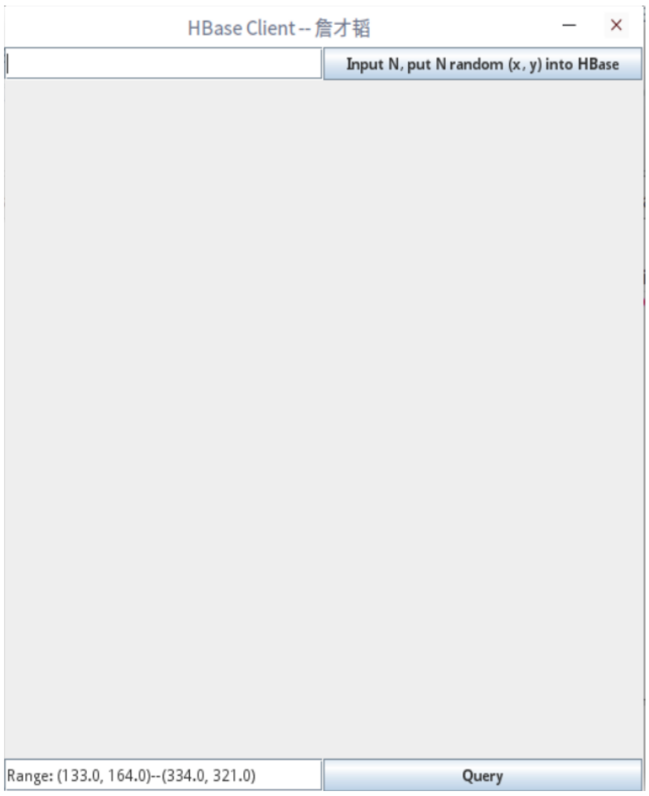

HBase Client -- 詹才韬
Input N, put N random (x, y) into HBase
Range: (133.0, 164.0)--(334.0, 321.0)
Query

图 1-1：Java 客户端界面

如图 1-1，Java 图形用户界面。

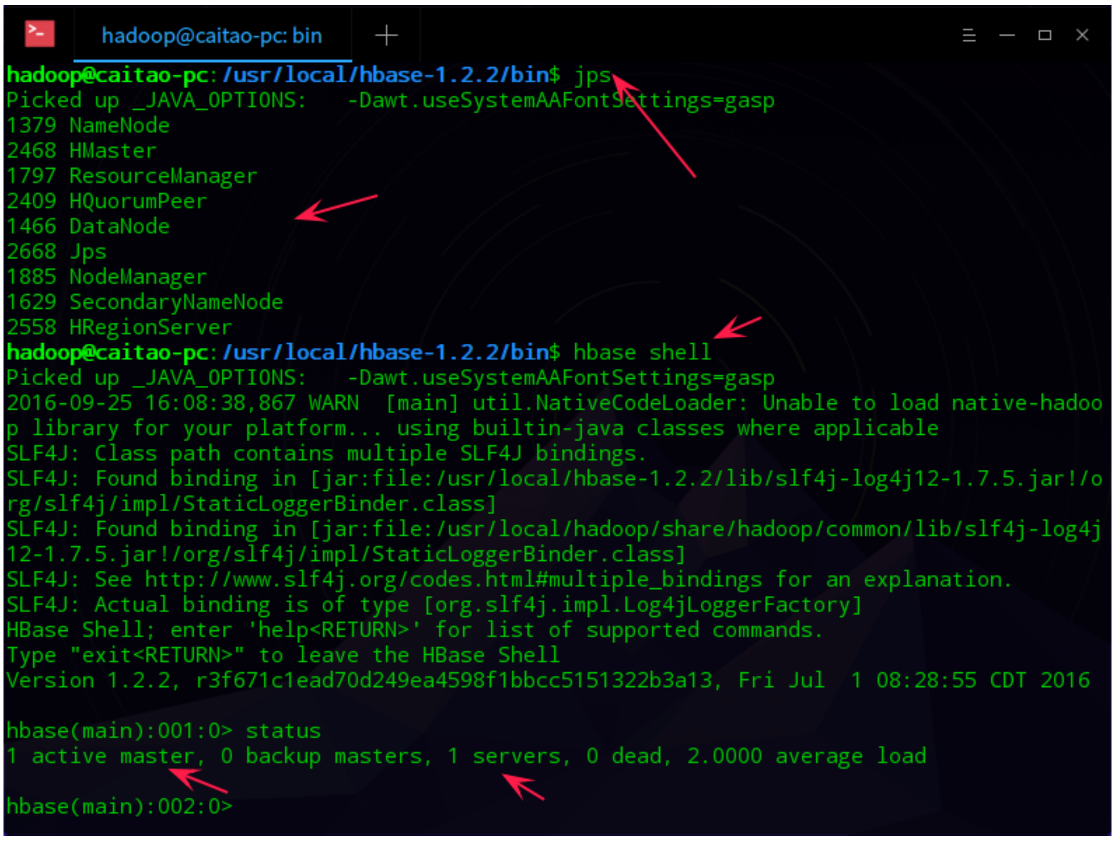
1. 左上角是一个输入框，用来输入一个数字，这个数字是往 HBase 中插入空间点的个数，按下右上角的按钮进行插入。
2. 中间一片大区域，是一个“正方形面板”，横着的是 X 坐标，取值范围[0, 512)，竖着的是 Y 坐标，取值范围[0, 512)。用鼠标的右键点击这个正方形面板，在左下角的输入框中会自动形成查询范围。
3. 左下角是一个输入框，用来输入查询范围。它可以源自点击正方形面板的自动生成自动生成，也可以手动输入调整。这个输入框的内容符合一个设定好的正则表达式，才可以进行查询。

1.2.2 后端：HBase 伪分布式

如图 1-2，在 linux 中输入 jps 命令(Java Virtual Machine Prosecc Status Tool)，查看当前在后台运行的 java 进程。从这里，可以看出 Hadoop 和 HBase 的伪分布式部署。

1. HDFS 相关进程：NameNode, SecondaryNameNode, DataNode
2. TARN 相关进程：ResourceManager, NodeManager
3. HBase 相关进程：HMaster, HRegionServer, HQuorumPeer

如图 1-2，输入 hbase shell 命令，可以直接和 HBase 进行交互。



```
hadoop@caitao-pc:/usr/local/hbase-1.2.2/bin$ jps
Picked up _JAVA_OPTIONS:   -Dawt.useSystemAAFontSettings=gasp
1379 NameNode
2468 HMaster
1797 ResourceManager
2409 HQuorumPeer
1466 DataNode
2668 Jps
1885 NodeManager
1629 SecondaryNameNode
2558 HRegionServer
hadoop@caitao-pc:/usr/local/hbase-1.2.2/bin$ hbase shell
Picked up _JAVA_OPTIONS:   -Dawt.useSystemAAFontSettings=gasp
2016-09-25 16:08:38,867 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoo
p library for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase-1.2.2/lib/slf4j-log4j12-1.7.5.jar!/o
rg/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j
12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.2, r3f671c1ead70d249ea4598f1bbcc5151322b3a13, Fri Jul  1 08:28:55 CDT 2016

hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load

hbase(main):002:0>
```

图 1-2：linux 后台运行图

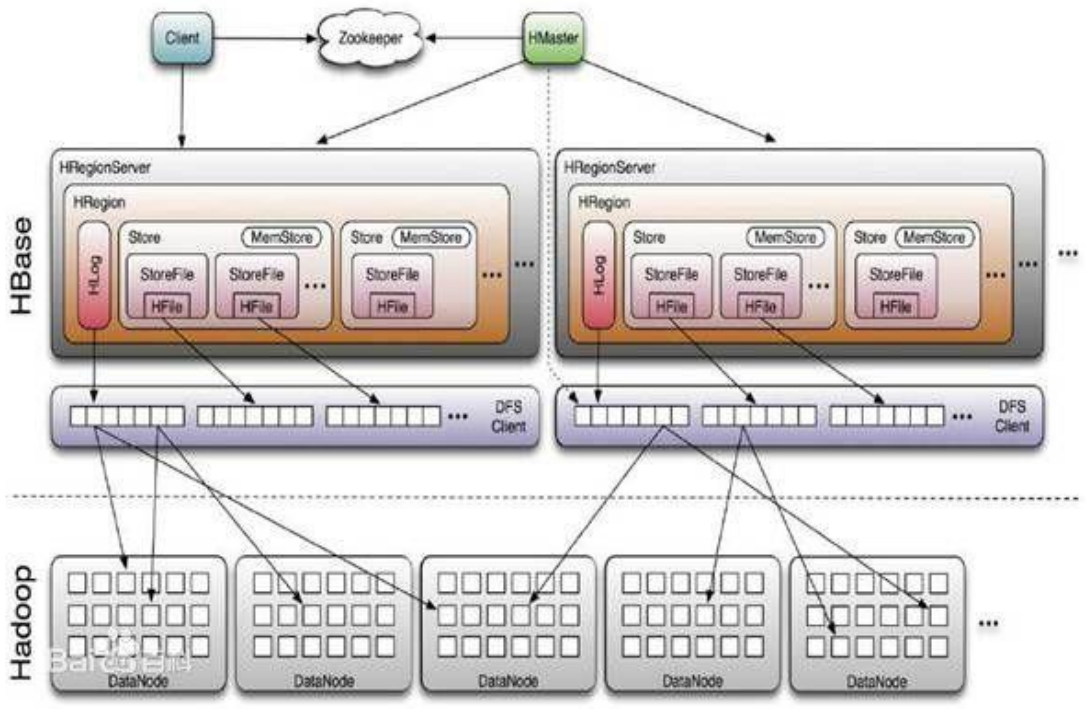


图 1-3：这个是 HBase 的架构，以及 HBase 和 Hadoop 的关系示意图

2 详细设计与系统实现

2.1 环境配置

2.1.1 配置步骤

配置是一个系统里面的重要，不可或缺的一个环节。比较繁琐，但是很重要。Hadoop 伪分布式部署需要配置四个文件：

1. core-site.xml
2. hdfs-site.xml
3. mapred-site.xml
4. yarn-site.xml

HBase 伪分布式配置需要配置：

1. hbase-site.xml
2. hbase-evn.sh

下面的配置，参考了[3], [4]

表 2-1：配置步骤

```
In this project, HBase runs in pseudo-distribution mode
Configurations are troublesome but indispensable.

//following are setups and configurations I did


//1.setups for Hadoop
sudo adduser hadoop
sudo usermod -G sudo hadoop
sudo apt-get install openssh-server
sudo apt-get install rsync

//install a jdk that can work for your hadoop's version if you don't have one.
sudo -l hadoop
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

//download hadoop-2.6.4 from apache's website, unzip, move to /usr/local/hadoop, chmod to 774
vim /home/hadoop/.bashrc    (add some paths at the end of .bashrc)
source ~/.bashrc


//2.configurations for Hadoop(pseudo-distribution mode)
cd /usr/local/hadoop/etc/hadoop
vim core-site.xml
vim hdfs-site.xml
vim mapred-site.xml
vim yarn-site.xml
vim hadoop-env.sh (export JAVA_HOME=${a jdk's path})
su -l hadoop
```

```
hadoop namenode -format

start-dfs.sh
start-yarn.sh

check http://localhost:8088    for ResourceManager web ui page
check http://localhost:50070 for HDFS web ui page


//3.a word count test
hadoop dfs -mkdir /user
hadoop dfs -mkdir /user/hadoop
hadoop dfs -mkdir /user/hadoop/input
hadoop dfs -put /etc/protocols /user/hadoop/input    (upload a file in linux file system to hdfs)

cd $HADOOP_INSTALL

bin/hadoop jar share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-2.6.4-sources.jar
org.ahache.hadoop.examples.WordCount input output

hadoop dfs -cat /user/hadoop/output/*    (check the results of word counting)

stop-dfs.sh
stop-yarn.sh


//4.setups and configurations for HBase(pseudo-distribution mode)
//download HBase-1.2.2 from apache's web
vim hbase-site.xml
vim ${HBASE_HOME}/conf/hbase-env.sh


//5.start HBase
start-dfs.sh
start-yarn.sh
start-hbase.sh

jps    // (NameNode + SecondaryNameNode + DataNode) + (ResourceManager + NodeManager) + (HMaster +
HRegionServer + HQuorumPeer)

hbase shell

create 'testTable','cf'
...
...


//6.the end
stop-hbase.sh
stop-dfs.sh
stop-yarn.sh
```

2.1.1 配置细节

表 2-2：Hadoop 和 HBase 的环境变量

```
#HADOOP START
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP END

#HBASE START
export HBASE_HOME=/usr/local/hbase-1.2.2
export PATH=$PATH:/usr/local/hbase-1.2.2/bin
#HBASE END
```

表 2-3：core-site.xml。fs.default.name 描述集群中 NameNode 结点的 URI，即协议：主机名：端口号；hadoop.tmp.dir 是 hadoop 文件系统一来的基础配置

```
<configuration>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://localhost:9000</value>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/hadoop/tmp</value>
        </property>
</configuration>
```

表 2-4：hdfs-site.xml。dfs.replication 决定系统里面文件块的数据备份个数

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
</configuration>
```

表 2-5：mapred-site.xml。

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

表 2-6：yarn-site.xml。yarn.nodemanager.aux-services 可以用户自定义一些服务

```
<configuration>
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
                <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
</configuration>
```

表 2-7：hbase_env.sh。配置环境变量。修改 jdk 路径。HBASE_MANAGES_ZK=true, 表示由 hbase 自己管理 zookeeper，不需要单独的 zookeeper

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HBASE_MANAGES_ZK=true
```

表 2-8：hbase-site.xml。hbase.rootdir 制定了 HRegion 服务器存放的位置，即数据存放的位置。hbase.cluster.distributed，HBase 的运行模式，

```
<configuration>
        <property>
                <name>hbase.rootdir</name>
                <value>hdfs://localhost:9000/hbase</value>
        </property>
        <property>
                <name>hbase.cluster.distributed</name>
                <value>true</value>
        </property>
</configuration>
```

2.2 数据库设计

2.2.1 希尔伯特曲线

文献[5] , [6], [7], [8]中讲解了为什么需要空间索引，怎么做空间索引，空间编码的方式有哪一些，如何实现等等。

现在假设我们以 U 字形来访问区域。在每个象限中，我们同样以 U 字形来访问子象限，但是要调整好 U 字形的朝向使得和相邻的象限衔接起来。如果我们正确地组织了这些 U 字形的朝向，我们就能完全消除不连续性，不管我们选择了什么分辨率，都能连续地访问整个区域，可以在完全地探访了一个区域后才移动到下一个。这个方案不仅消除了不连续性，而且提高了总体的局域性。按照这个方案得到的图案就是希尔伯特曲线。

希尔伯特曲线属于一类被称为空间填充曲线的一维分形，因为它们虽然是一维的线，却可以填充固定区域的所有空间。Hilbert 曲线对于空间索引它们也是有用的，因为它们展现的正是我们需要的局域性和连续性。

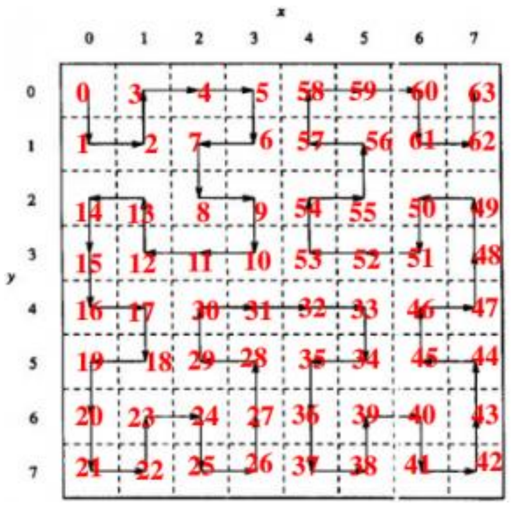


图 2-1：三阶希尔伯特曲线

如图 2-1，希尔伯特曲线的目的是降维：将二维的（x，y）降成一维的希尔伯特距离，进而转换成字符串。变成字符串之后，便可以此为基础设计行键，HBase 在该行键上做索引。

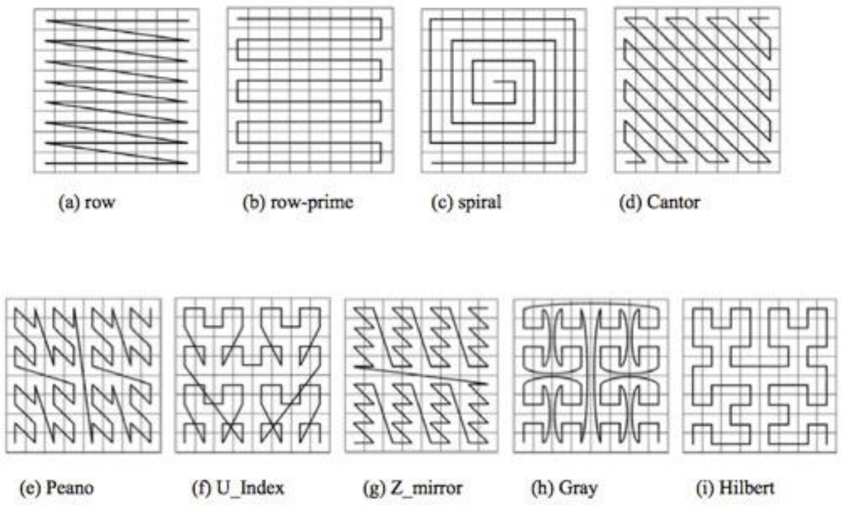


图 2-2：很多曲线填充编码方式。

图 2-2 中可视化了多种曲线填充方式。在本次项目中，采用的是最后一个 Hilbert。
图 2-3 中描述了 Hilbert 曲线中的 4 种基础变换，这 4 种变换不断组合，构成了 Hilber 曲线。通过观察发现，子象限的曲线是源曲线的简单变换。

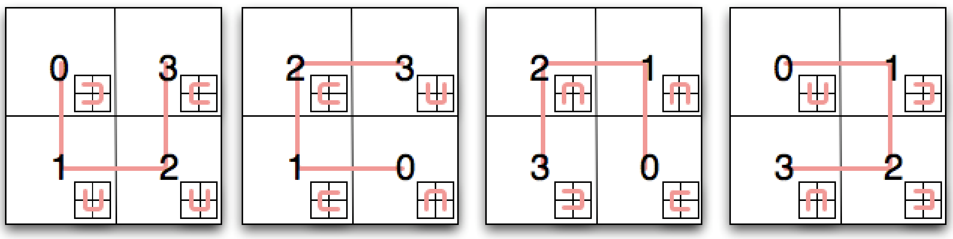


图 2-3：4 种变换。从左到右分别为 square “a”, square “b”, square “c”, square “d”。
每一个 square 里面又可以生成小 square。这样不断“递归下去”，类似于分形。

求解 Hilbert 曲线距离可以使用递归的方法，但是如果用循环的方法实现的话，程序运行效率更高，源代码如表 2-9 所示。

表 2-9：希尔伯特编码源码

```
public class HilbertCurve
{
        static ImmutableMap<String, Pair[][]> hilbertMap = null;
        static class Pair
        {
                int position;      // each square in Hilbert table has four positions
                String square;    // there are four squares in Hilbert Table
                Pair(int position, String square)
                {
                        this.position = position;
                        this.square = square;
                }
        }
        static
        {
                hilbertMap = ImmutableMap.of(
"a", new Pair[][] { { new Pair(0, "d"), new Pair(3, "b") }, { new Pair(1, "a"), new Pair(2, "a") } },
"b", new Pair[][] { { new Pair(2, "b"), new Pair(3, "a") }, { new Pair(1, "b"), new Pair(0, "c") } },
"c", new Pair[][] { { new Pair(2, "c"), new Pair(1, "c") }, { new Pair(3, "d"), new Pair(0, "b") } },
"d", new Pair[][] { { new Pair(0, "a"), new Pair(1, "d") }, { new Pair(3, "c"), new Pair(2, "d") } });
```

```
        }
        /**
          * dimensionality reduction
          * @param x
          * @param y
          * @param order, an n order curve fills a (2^n)*(2^n) grid
          * @return a string, which is encoded from (x, y) using Hilbert curve
          */
        public static long encode(int x, int y, int order)
        {
                String currentSquare = "a";
                long position = 0;    // max = 2^63-1
                int squareX = 0;
                int squareY = 0;
                int squarePosition = 0;
                for(int i = order - 1; i >= 0; i--)
                {
                        position <<= 2;
                        squareX = (x & (1 << i)) > 0 ? 1 : 0;
                        squareY = (y & (1 << i)) > 0 ? 1 : 0;
                        Pair pair = hilbertMap.get(currentSquare)[squareY][squareX];

                        squarePosition = pair.position;
                        currentSquare = pair.square;
                        position |= squarePosition;
                }
                return position;
        }
        public static void main(String[] args)
        {
                System.out.println(encode(5, 2, 3));
        }
}
```

2.2.2 行键设计

Rowkey: 长度等于 16 字节

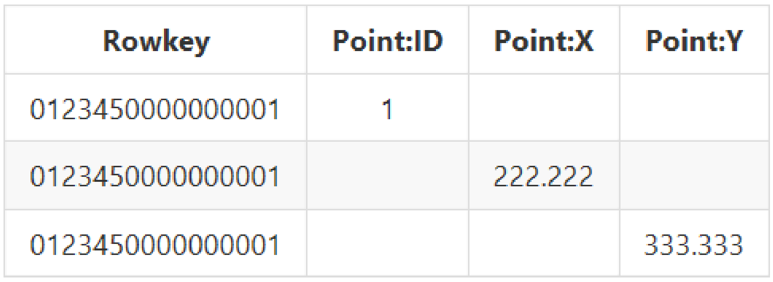
1. 假设有一个空间二维点(x0, y0)，通过希尔伯特曲线运算得到希尔伯特距离，这个距离转换成字符串，即为编码 *HilbertCode*。在本次项目中，希尔伯特曲线的 order 等于 9，编码的长度等于 6 字节。
2. 给点(x0, y0)赋予一个 ID 号，转换为长度为 10 的字符串 *ID*。
3. Rowkey = *HilbertCode* + *ID*，拼接起来，长度为 16 字节。

表 2-10：计算行键。如果一个点(x0, y0)的希尔伯特距离为 12345，ID 号为 1。那么行键等于 0123450000000001

```
/**
  * generate a Point's row key. row key = hilbert + id
  * @param point, a point that needs to generate a row key
  * @return point's row key
  */
public static String generatePointRowKey(Point point)
{
        int order = 9;    // x, y : (0, 2^order-1)
        StringBuffer rowKey = new StringBuffer();
        int x = (int) point.getX();
        int y = (int) point.getY();
        int id = point.getID();

        long encode = HilbertCurve.encode(x, y, order);
        String encodeStr = String.valueOf(encode);
        String idString = String.valueOf(id);
        byte[] temp = new byte[hilbertLength - encodeStr.length()];
        Arrays.fill(temp, (byte) '0');
        rowKey.append(new String(temp));
        rowKey.append(encodeStr);

        temp = new byte[idLength - idString.length()];
        Arrays.fill(temp, (byte) '0');
        rowKey.append(new String(temp));
        rowKey.append(idString);

        return rowKey.toString();
}
```

2.2.3 HBase schema

存储模型参考了文献[9], [10], [11], [12]



Rowkey	Point:ID	Point:X	Point:Y
0123450000000001	1		
0123450000000001		222.222	
0123450000000001			333.333

图 2-4：HBase 的列簇，qualifier 列

如图 2-4。假设有一个空间二维点（222.222, 333.333），它的 ID 号等于 1。那么它插入 HBase 中需要插入三次。列簇=Point，三个列(qualifier)分别为 ID, X, Y。

2.3 类设计 UML 图

1. Point，封装一个空间二维点
2. QueryRect，封装查询范围。范围由两个点组成
3. ClientApp，封装客户端界面的所有空间和操作
4. HBase，封装 HBase 的核心 API
5. HilberCurve，封装希尔伯特曲线算法

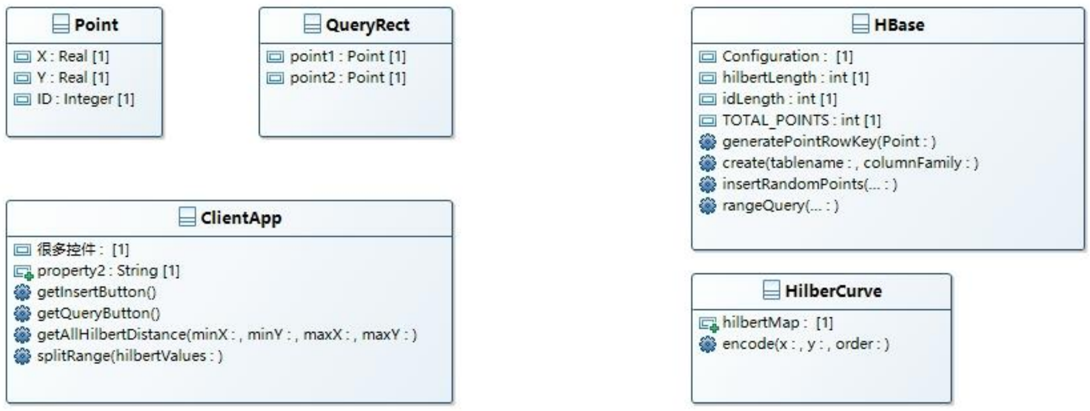


图 2-5：整个项目一共五个类

2.4 插入算法

2.4.1 Java 客户端

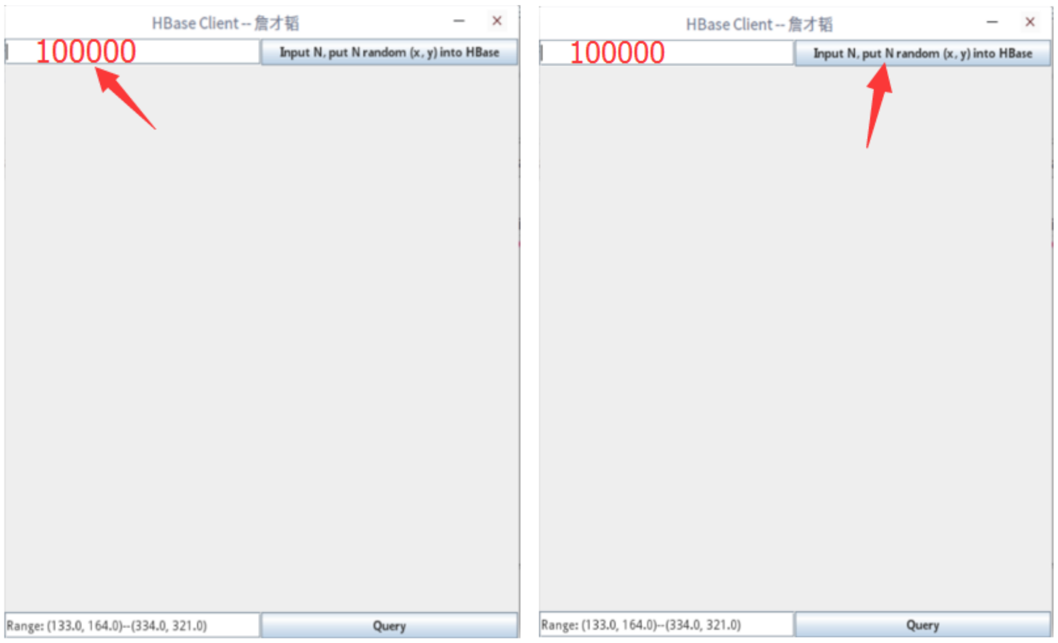

HBase Client -- 詹才韬
100000
Input N, put N random (x, y) into HBase
Range: (133.0, 164.0)--(334.0, 321.0)
Query
HBase Client -- 詹才韬
100000
Input N, put N random (x, y) into HBase
Range: (133.0, 164.0)--(334.0, 321.0)
Query

图 2-6：在客户端左上角输入一个数字，代表插入若干点。

按下按钮，进入回掉函数，按钮里面有在监听。

插入流程：

1. 在左上角的文本框输入要插入点的个数
2. 按下“Input N, put N random (x, y) into HBase”按钮
3. 按钮的监听器调用 HBase 的 API，传入相关参数

表 2-11：按下插入按钮，进入按钮的监听器，在监听器里面调用 HBase 的 API

```
private JButton getInsertButton()
{//  当第一次调用这个方法的时候，rButton == null，进行初始化操作
        if (insertButton == null)
        {
                insertButton = new JButton("Input N, put N random (x, y) into HBase");
                insertButton.addActionListener((ActionEvent e) ->
                {
                        try
                        {
                                String string = editInsertNumber.getText();
                                int insertNum = Integer.valueOf(string);

                                caitaoHBase.insertRandomPoints(insertNum, MAX_X, MAX_Y,
                TABLE_NAME, COLUMN_FAMILY, QUALIFY_X, QUALIFY_Y, QUALIFY_ID);

                                System.out.println("Successfully inserted " + insertNum +" random points");
                        }
                        catch (NumberFormatException exception)
                        {
                                System.err.println("didn't input an integer");
                                exception.printStackTrace();
                        }
                        catch (Exception exception)
                        {
```

```
                        exception.printStackTrace();
                    }
                });
        }
        return insertButton;
}
```

2.4.2 HBase 的 API

HBase 没有类似于关系型数据库的 SQL 语句。要对数据库进行增、删、改、查等操作，需要调用 HBase 提供的 API 函数，如下：

1. 接收 Java 客户端传来的参数
2. 默认使用{projectDirectory}/conf/hbase-site.xml 里面的配置，创建一个 Connection
3. 使用 Connection 创建一个 Table
4. 创建一个 List<Row> batch，用来进行批处理
5. 不断随机生成空间二维的点，把这些点添加到 Put 实例中，Put 实例加入到 batch 中
6. 一次性把 batch 里面的所有东西插入 table 中

表 2-12：向 HBase 插入点

```
public void insertRandomPoints(int insertNum, int maxX, int maxY, String tableName,
                        String columnFamily, String qualifyX, String qualifyY, String qualifyID) throws IOException
{
        Connection connection = ConnectionFactory.createConnection(cfg);
        Table table = connection.getTable(TableName.valueOf(tableName));
        List<Row> batch = new ArrayList<Row>();    // a batch of puts is faster than many seperate puts
        for(int i = 0; i < insertNum; i++)
        {
                Point point = new Point(++TOTAL_POINTS);
                point.random(maxX, maxY);
                String rowKey = generatePointRowKey(point);
                Put put = new Put(Bytes.toBytes(rowKey));
         put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyID), Bytes.toBytes(point.getStringID()));
         put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyX), Bytes.toBytes(double2String(point.getX())));
         put.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyY), Bytes.toBytes(double2String(point.getY())));
                batch.add(put);
        }
        Object[] results = new Object[batch.size()];
        try
        {
                table.batch(batch, results);
        }
        catch(Exception e)
        {
                System.err.println("Error: " + e);
        }
        table.close();
        connection.close();
}
```

2.5 查询算法

2.5.1 Java 客户端

查询的流程如下：

1. 鼠标右键点击两次，中间的面板捕获坐标，假设分别点击的两个坐标分别为(116.0, 134.0)和(284.0, 267.0)
2. 按照固定的模式自动生成查询语句，查询的范围是(116.0, 134.0)--(284.0, 267.0)。这是一个矩形，左上角坐标和右下角坐标自动调整为这个矩形相应的顶点
3. 按下查询按钮，正则表达式捕获查询范围的四个范围边界数字。
4. 计算 hilber 曲线距离，排序，分段
5. 调用 HBase 的 API 函数，传入相关参数

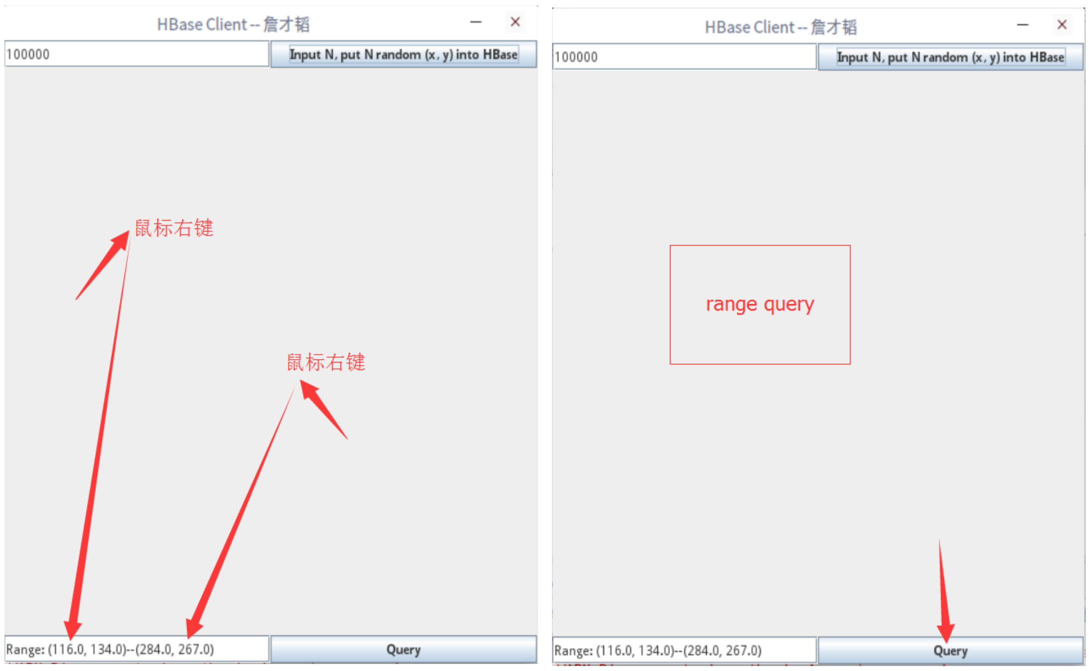

HBase Client -- 詹才韬
100000
Input N, put N random (x, y) into HBase
鼠标右键
鼠标右键
Range: (116.0, 134.0)--(284.0, 267.0)
Query
HBase Client -- 詹才韬
100000
Input N, put N random (x, y) into HBase
range query
Range: (116.0, 134.0)--(284.0, 267.0)
Query

图 2-7：Java 客户端查询界面

2.5.2 HBase 的 API

HBase 没有类似于关系型数据库的 SQL 语句。要对数据库进行增、删、改、查等操作，需要调用 HBase 提供的 API 函数，如下：

1. 接收 Java 客户端传来的参数
2. 默认使用{projectDirectory}/conf/hbase-site.xml 里面的配置，创建一个 Connection
3. 使用 Connection 创建一个 Table
4. 创建一个 FilterList，在 FilterList 里面添加四个 SingleColumnValueFilter，以传入的 minX, minY, maxX, maxY 作为精确过滤的边界
5. 依据扫描范围（传进来的参数中），和 FilterList 一段一段扫面 HBase 的表，获取数据，保存在临时的对象
6. 把临时保存的所有数据返回给 Java 客户端

表 2-13：在 HBase 里查询数据

```
public ArrayList<Point> rangeQuery(String tableName, String columnFamily, String qualifyX, String qualifyY,
                                    long[] hilbert, ArrayList<Integer> ranges, double rectMinX, double
                              rectMinY, double rectMaxX, double rectMaxY) throws IOException
{
        Connection connection = ConnectionFactory.createConnection(cfg);
        Table table = connection.getTable(TableName.valueOf(tableName));
        ArrayList<Point> queriedPoints = new ArrayList<Point>();

        FilterList filterList = new FilterList(Operator.MUST_PASS_ALL);
        SingleColumnValueFilter filterMinX = new SingleColumnValueFilter(
                        Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyX),
                        CompareFilter.CompareOp.GREATER_OR_EQUAL,
                        new BinaryComparator(Bytes.toBytes(double2String(rectMinX))));
        filterList.addFilter(filterMinX);

        SingleColumnValueFilter filterMaxX = new SingleColumnValueFilter(
                        Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyX),
                        CompareFilter.CompareOp.LESS,
                        new BinaryComparator(Bytes.toBytes(double2String(rectMaxX))));
        filterList.addFilter(filterMaxX);

        SingleColumnValueFilter filterMinY = new SingleColumnValueFilter(
                        Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyY),
                        CompareFilter.CompareOp.GREATER_OR_EQUAL,
                        new BinaryComparator(Bytes.toBytes(double2String(rectMinY))));
        filterList.addFilter(filterMinY);

        SingleColumnValueFilter filterMaxY = new SingleColumnValueFilter(
                        Bytes.toBytes(columnFamily), Bytes.toBytes(qualifyY),
                        CompareFilter.CompareOp.LESS,
                        new BinaryComparator(Bytes.toBytes(double2String(rectMaxY))));
        filterList.addFilter(filterMaxY);

        for(int i = 0, j = 1; j < ranges.size(); i+=2, j+=2)
        {
                String startStr = String.valueOf((hilbert[ranges.get(i).intValue()]));
                String endStr     = String.valueOf((hilbert[ranges.get(j).intValue()]) + 1);    // [startRow, endRow]

                StringBuffer startBuffer = new StringBuffer();
                byte[] temp = new byte[hilbertLength - startStr.length()];
                Arrays.fill(temp, (byte)'0');
                startBuffer.append(new String(temp));
                startBuffer.append(startStr);

                StringBuffer endBuffer     = new StringBuffer();
                temp = new byte[hilbertLength - endStr.length()];
                Arrays.fill(temp, (byte)'0');
                endBuffer.append(new String(temp));
                endBuffer.append(endStr);

                Scan scan = new Scan();
                scan.setStartRow(Bytes.toBytes(startBuffer.toString()));
                scan.setStopRow(Bytes.toBytes(endBuffer.toString()));
                scan.setFilter(filterList);

                ResultScanner scanner = table.getScanner(scan);
                for(Result result : scanner)
                {
                        Point point = new Point();
                        int k = 0;
                        for(Cell cell : result.rawCells())
                        {
                                if(k == 1) // not beautiful code
                                {
            double x = Double.parseDouble(Bytes.toString(cell.getValueArray(), cell.getValueOffset(), cell.getValueLength()));
```

```
                        point.setX(x);
                    }
                    if(k == 2)
                    {
    double y = Double.parseDouble(Bytes.toString(cell.getValueArray(), cell.getValueOffset(), cell.getValueLength()));
                        point.setY(y);
                    }
                    ++k;
                }
                queriedPoints.add(point);
            }
        }
        double ratio = (rectMaxX - rectMinX) * (rectMaxY - rectMinY) / (512*512);
        System.out.println("theoretical points = " + (int)(TOTAL_POINTS * ratio) + "\nreality points = " + queriedPoints.size());
        return queriedPoints;
    }
```

3 系统测试

3.1 测试

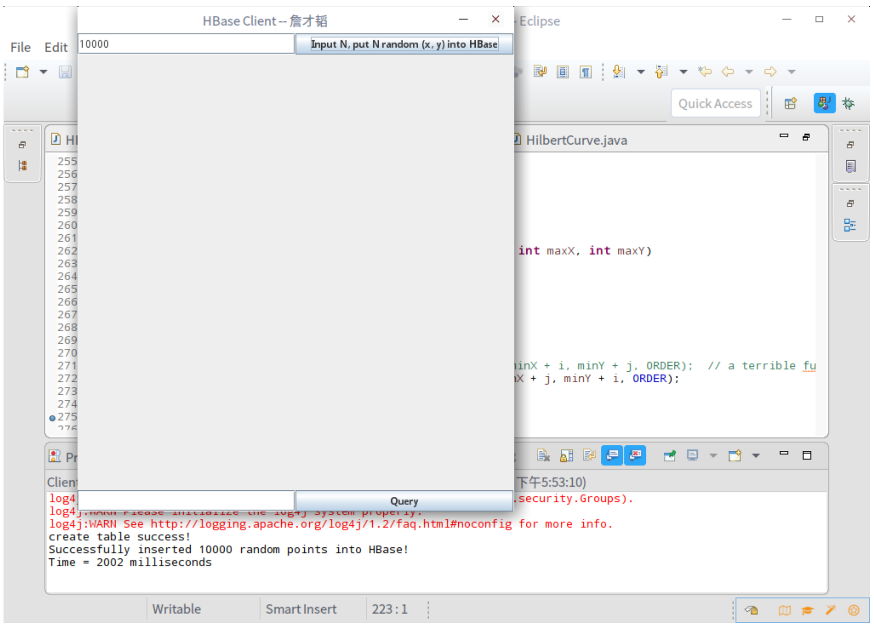
3.1.1 插入测试

程序日志如下：

表 3-1：创建表和插入日志

```
create table success!
Successfully inserted 10000 random points into HBase!
Time = 2002 milliseconds
```

插入的点是随机分布的。

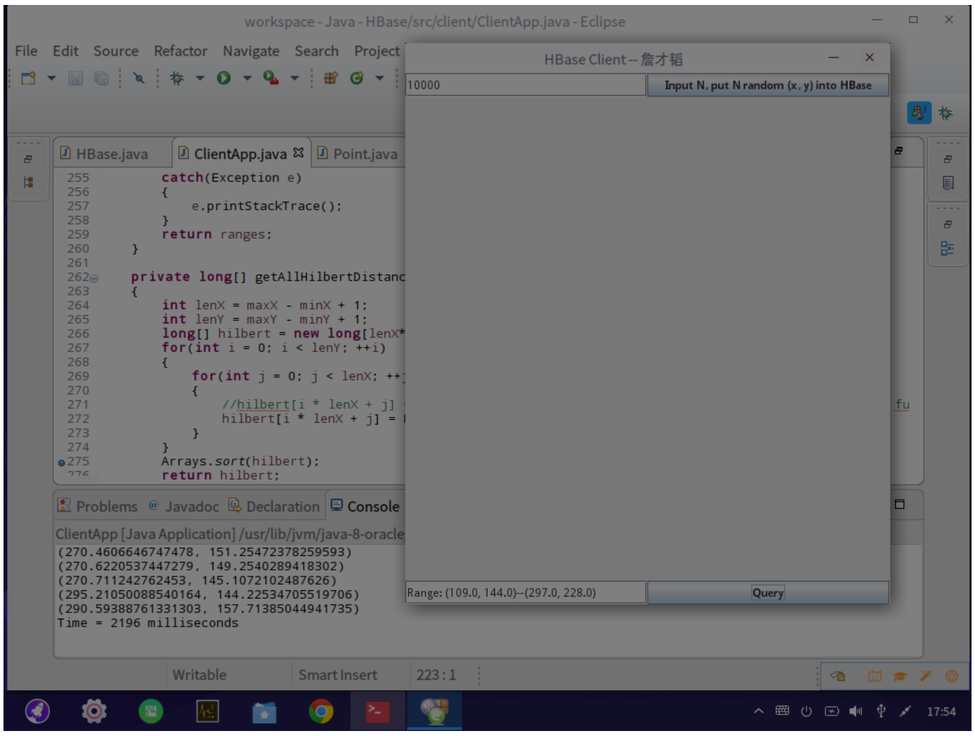


```
int maxX, int maxY)
inX + i, minY + j, ORDER);  // a terrible fu
X + j, minY + i, ORDER);
```

```
下午5:53:10)
log4                                          .security.Groups).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
create table success!
Successfully inserted 10000 random points into HBase!
Time = 2002 milliseconds
```

图 3-1：成功插入 10000 个随机空间点

3.2.2 查询测试



```java
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return ranges;
    }

    private long[] getAllHilbertDistanc
    {
        int lenX = maxX - minX + 1;
        int lenY = maxY - minY + 1;
        long[] hilbert = new long[lenX*
        for(int i = 0; i < lenY; ++i)
        {
            for(int j = 0; j < lenX; ++j
            {
                //hilbert[i * lenX + j]
                hilbert[i * lenX + j] = 
            }
        }
        Arrays.sort(hilbert);
        return hilbert;
```

```
ClientApp [Java Application] /usr/lib/jvm/java-8-oracle
(270.4606646747478, 151.25472378259593)
(270.6220537447279, 149.2540289418302)
(270.711242762453, 145.1072102487626)
(295.21050088540164, 144.22534705519706)
(290.59388761331303, 157.71385044941735)
Time = 2196 milliseconds
```

图 3-2：查询范围(109.0, 144.0) – (297.0, 228.0)，查询花了 2196 毫秒

1. 查询返回 589 个点。在范围(109.0, 144.0) – (297.0, 228.0)，总共有 589 个点。理论上有 $10000 \times 188.0 \times 84.0 / _{512 \times 512} = 602$ 个点。实际值与理论只相差 2%，误差很小。理论计算的前提条件是所有的空间坐标随机分布。
2. 查询时间为 2196 毫秒

表 3-2：查询返回的结果。范围：(109.0, 144.0) – (297.0, 228.0)

theoretical points = 602	(218.77243316673793, 169.57575374804998)	(122.84194151589475, 194.45196859467427)
reality points = 589	(219.6729656365685, 168.78754842244626)	(124.79641817954894, 198.05424914887737)
(145.57706767135778, 144.2146101451351)	(222.67582833222286, 167.36794097270302)	(125.07396618521398, 192.4219279142024)
(147.94142046541253, 150.2306132102836)	(219.16136957615134, 163.77525413941703)	(126.9936723120336, 192.92086911879596)
(146.42997829350105, 148.48713508486787)	(214.13073102757454, 175.37678103241996)	(126.58396456372583, 193.17223074735762)
(150.35889455750527, 148.83518738321015)	(214.95871940030088, 175.860474257372)	(127.31868584488325, 193.58344021601158)
(149.09638763038964, 146.0937282164041)	(206.72431448899022, 170.48637351868598)	(124.60764317976651, 191.179482668145)
(159.12000653765722, 145.455001130036)	(207.47618721997173, 164.62029511442347)	(125.40414251803804, 187.41514224027247)
(155.12553520473102, 148.81290654053493)	(201.26058780136935, 166.73708064463676)	(126.88513969868922, 187.52743579935998)
(156.06083622591456, 152.32951537811329)	(197.94532278539447, 162.4831094495521)	(126.3309121387826, 179.49244641374077)
(159.43848568461533, 158.94467733734132)	(195.78876567695733, 167.53904087701522)	(118.4913716816373, 182.49058497582507)
(153.7839542363892, 159.48984899021218)	(194.15822263937605, 168.84115880296912)	(116.10424669551327, 178.45113092263716)
(145.645673705165, 159.87122210905795)	(192.4179234401151, 173.41164645239746)	(115.59192561333384, 186.9655144803346)
(140.6889068580466, 158.62197127403277)	(193.07326811039746, 174.3007461566969)	(109.04526575145655, 188.67595636830453)
(138.68934145713075, 158.3535434268607)	(197.62672760147188, 179.52310184000913)	(109.14051540759726, 172.79361716260246)
(139.57112207826583, 152.66702970136123)	(201.84401184350878, 178.84435358644038)	(111.58750109273387, 169.0888091609536)
(142.49040934497566, 150.60503175042703)	(204.2148225698889, 179.35146288369918)	(109.08533418182128, 169.80036055373995)
(140.66536223801842, 151.49309986984986)	(202.72414483884882, 180.40313041505084)	(111.81198437964946, 165.39539627019957)
(139.3246064571747, 149.45786295177788)	(200.98247274814037, 184.16662966126967)	(109.00649535783663, 160.9902059918782)
(139.70710662283983, 151.7060627269231)	(205.35494405748773, 189.67373862512528)	(114.17972413614035, 168.09548289994586)
(135.92640191211075, 151.95101537350735)	(200.00193655749865, 188.83127433056382)	(116.25635660332591, 171.6619743536403)
(128.6253671624118, 146.49097878296294)	(197.93387940787346, 191.8184981638745)	(125.86554581419091, 172.05673926241792)
(128.79735913031442, 153.84169678078314)	(195.64382293747667, 191.7146245146683)	(125.25555580893541, 170.3519710303588)
(131.3875299902565, 156.08506636503375)	(195.31254595396763, 194.09078693387357)	(125.6725808721215, 169.46090023626903)

(129.59783939923767, 157.8350457108623)	(192.99402895201007, 194.21100170412598)	(126.72626746918257, 169.1197714628467)
(130.34903353204805, 167.37966012319612)	(195.35091859809978, 198.31446556467927)	(123.15296756719903, 155.19898045549388)
(132.02848566745706, 167.26433772312294)	(202.4584971228403, 194.94475129584282)	(122.53888863282731, 158.72366480798286)
(142.94756593574851, 164.6925618060382)	(203.89933809004947, 194.63914122870847)	(121.34951437710708, 158.90120856578307)
(136.31760975718765, 168.63643765636363)	(201.4410627370549, 201.39390063774073)	(118.71268174508987, 159.2671328242365)
(140.81694541154178, 169.6772187824534)	(207.82893025933885, 200.7269993904692)	(117.06539403987051, 158.04305650344156)
(132.48979734812548, 170.67956698383495)	(202.60431508393486, 206.6421536201957)	(113.61326545169828, 155.18017388830106)
(131.84412873867035, 191.2240753823586)	(197.23173006787118, 200.5646356643349)	(119.42853277278903, 154.43206347328743)
(132.43813278453746, 185.20765536165453)	(194.5655777113002, 205.9801091868854)	(112.9636194603616, 147.0974282611462)
(141.84968208889205, 188.68714836670966)	(197.44617689921876, 220.87675580197407)	(112.31147810972186, 146.98708468160504)
(143.82808483929256, 186.9243606016908)	(196.04528256369628, 217.2183370094047)	(113.07911317690622, 146.02046444293893)
(138.1562863669589, 182.27260159912657)	(204.8844104997694, 214.65752010892663)	(122.65758439836367, 147.47903619322813)
(138.3321884812953, 178.49678529435948)	(203.6077383613357, 211.5373089514406)	(121.77349369102069, 148.1900553844382)
(141.23613260364937, 177.2328650485071)	(212.19876555439203, 208.87370981251536)	(110.30345175659215, 144.7664133219858)
(146.08217663635912, 189.4343696188268)	(214.05911414958672, 208.2933788160605)	(110.57841013346285, 150.69190112147476)
(148.70583584403704, 189.1565815097681)	(215.07069229814653, 212.0440021625758)	(109.43869307399154, 153.3367139342011)
(153.29153097353003, 185.05851279190722)	(212.90436412098433, 219.57776950554023)	(110.31825892670855, 154.11641265080402)
(157.86670113751853, 189.92832020583649)	(221.7969749387508, 220.19391171350026)	(109.3667456073909, 155.09074367970146)
(158.24925801136823, 190.12620164624116)	(222.9446435200245, 222.72453029115553)	(109.89456729607639, 155.95692353790037)
(157.74981973866238, 181.41166461922222)	(222.43399461141007, 217.0653727597794)	(296.82714274593036, 227.57113873130328)
(157.03994874002296, 174.19211235345261)	(223.55988052453444, 212.41769419609432)	(259.81009335871795, 224.79205044026094)
(159.20157551139494, 170.6213949661601)	(216.8337825886817, 214.08518139250873)	(263.4090656702711, 224.23592136304705)
(152.71809373573194, 168.5462124098413)	(216.48536903993215, 213.73711604040523)	(265.9633145424744, 227.38280484172662)
(147.81290665400047, 168.87304952590512)	(222.9117499686858, 205.0860472465112)	(269.11652884372194, 227.67186067611595)
(149.4665796951943, 170.03566036562955)	(222.9457077627448, 205.76335428856993)	(273.2827433033796, 227.14091674930273)
(148.15672501129058, 167.00409157679178)	(218.8468669468151, 203.23048088262686)	(279.6375021803322, 224.37335472174004)
(147.5052386874773, 163.15065912886087)	(215.6446791965014, 204.92000014193434)	(283.2770062712431, 225.2403193527909)
(154.8403733262952, 162.99620111885372)	(208.0900127788342, 203.04278350402774)	(284.92902748847933, 225.46780677435856)
(159.78864435380854, 164.04144087339648)	(209.28661187329152, 193.40664117232046)	(285.82466593380127, 215.8403495276068)
(158.26934655612416, 162.29826751878727)	(218.56512867012094, 193.99223840905455)	(280.6033460582035, 209.35852277046303)
(158.8806038258852, 163.40318790709034)	(216.11503636392905, 195.90245590553542)	(275.54052724140587, 212.6592437084925)
(167.45349299812966, 162.74412975763096)	(219.91217239925896, 199.09782116245123)	(273.0493189882589, 217.26542660317)
(165.30652338773348, 163.9620086842375)	(223.037841588579, 192.90315168054963)	(273.6255542816021, 223.55465290923183)
(173.7896929399439, 163.06948482001695)	(225.11633563472532, 194.71597231632325)	(263.8609924978871, 223.51225263310914)
(174.41475174962324, 171.62835875975344)	(229.09060068929756, 195.06861939499737)	(259.6272272775659, 216.46180937689002)
(168.6079761661598, 173.9286157399377)	(224.89110330043053, 202.25542062332528)	(263.5436692857124, 216.16344353664834)
(169.5872046805685, 174.82153448702348)	(231.98883170692528, 207.99156670783822)	(263.21347361451194, 213.29308791826293)
(167.92116643203008, 171.38790056625083)	(233.66236686959917, 200.27183876553102)	(260.85233079347324, 213.49845270453613)
(165.05909851944398, 170.42427751283856)	(234.8015013457058, 205.68782240156509)	(259.3385748018914, 209.05615119203185)
(163.0356085226652, 172.95549794252975)	(235.82751766837998, 204.19822964962293)	(266.6355602354942, 212.9174183883094)
(162.07450931555962, 175.2210736437765)	(236.85441988919297, 204.90040124673453)	(270.47430359869645, 214.89737504721234)
(163.69117655166355, 178.28988694376739)	(238.77652759134037, 199.42091176945576)	(271.6853410630483, 215.99068171838013)
(164.8132385821453, 176.5793276661709)	(232.4029838140026, 198.94106323431708)	(270.0063390831768, 202.3392732507756)
(161.4782732615215, 180.7398634979998)	(241.81685223733712, 199.87399483019584)	(267.9526701618095, 203.29777681397405)
(160.68148826636872, 184.73326159976278)	(249.5894871762374, 194.31417439854596)	(262.94045780109195, 206.09721598899307)
(165.40950965929773, 188.1610621367825)	(248.6347491506748, 199.78774933532287)	(263.2661099070115, 204.02973214700484)
(174.81298333562017, 181.78072222371935)	(255.37511580655985, 205.74719229496014)	(260.38801056105626, 204.9735106914763)
(173.14555665166716, 177.2021500981141)	(247.1186161278389, 207.73369085975145)	(256.881293241473, 204.4935881960854)
(174.77930094367986, 179.991801801394)	(240.67128547068535, 204.37493387393323)	(261.58094400206664, 202.432753651307)
(182.74778884252117, 177.42524311594713)	(245.8258845595103, 210.15808429409236)	(256.12896320445856, 197.70675796821945)
(183.44815287180302, 179.10216595666083)	(244.93132194474367, 208.89003512808233)	(264.6156750605905, 199.4575541616838)
(179.35981673084575, 189.55194427392638)	(251.1921120385203, 213.00226181589693)	(266.78132293382515, 196.8567571609542)
(187.62078423877784, 190.36287093630432)	(249.65220630532093, 212.6060416204527)	(269.352056554978, 197.52551958431076)
(189.16759933985583, 188.92867241274638)	(246.35610127688994, 222.10050581443016)	(269.85135607466697, 198.7830581758323)
(185.69738982324537, 183.68216532261897)	(247.1907102991571, 218.49146078009846)	(272.340897409287, 194.1304866769151)
(187.42145985720907, 180.75222846172363)	(243.13946920363003, 223.34594500198892)	(275.2111142752175, 192.1491737965993)
(188.25003561403253, 176.36049210419657)	(235.93378285984033, 223.08506237740028)	(279.1623254371254, 194.25572423273735)
(190.3090430674594, 177.8865766119548)	(233.65908076074567, 210.49393950927487)	(277.2899790145993, 194.14153468628348)
(180.5129344742794, 170.63264741316914)	(233.8172391542554, 211.0734653428574)	(275.80601905013725, 202.6032307794851)
(181.58842448681145, 167.46079938306457)	(232.55819250171982, 213.06066668221104)	(272.5547639876581, 207.72984934369754)
(176.81931995076326, 163.44804381115028)	(232.65895644928418, 215.32042788634635)	(274.2677590492004, 205.60199853969726)
(179.23946178611533, 161.15534790221642)	(228.01981988787048, 218.91576310817754)	(286.38572254445114, 196.12055850429573)
(179.3665374894776, 160.47283126123932)	(228.43901897889037, 219.74792260062975)	(284.18074765621805, 192.9567088162898)
(186.26129427459125, 162.9585140305361)	(235.74443648583963, 226.5868497445216)	(287.92937507322955, 193.62250684508257)
(188.7212942177557, 166.9596112592845)	(243.2234331779149, 227.62620606533142)	(289.0778186939552, 199.4364835519462)
(190.24630851602706, 166.97387689257727)	(218.26632770099775, 227.83277943812357)	(296.4620813622642, 193.90973300855336)
(190.95202477721102, 165.08468819193837)	(211.0254250062074, 227.97095997715724)	(296.58746321847667, 205.09051937687224)
(191.1951536675095, 149.28434252054342)	(207.30250283682756, 227.31751424820152)	(296.7865446266014, 206.9102175042371)
(185.42948910174982, 147.0055937110492)	(195.44312730376862, 225.43761882103558)	(295.0224936007477, 204.17139924218884)
(181.90854730362184, 144.40419365897253)	(194.95948161856256, 224.36696127879986)	(292.37688742705393, 200.95616804661813)
(176.34021896775323, 151.80416108183266)	(193.98734264887594, 225.61588492599816)	(292.0444452332515, 200.64237437274204)
(175.25008843769632, 157.4578581609187)	(160.85422439159402, 225.427775277794)	(290.06575655564967, 211.078453734447)

(170.70210934693864, 156.39800271968272)	(167.73330689762548, 225.98264659040217)	(294.45203157962715, 209.70324415540915)
(170.8100424059911, 156.19468535473555)	(172.0640966524013, 227.71977787874238)	(288.27006403691405, 213.04937422568014)
(168.59904694017166, 159.73422367162914)	(187.08819004494023, 224.997994880816)	(290.60077082740924, 220.0605233488273)
(161.23288513668268, 159.40594882822109)	(188.4067395754057, 222.61283326243284)	(292.36538399862576, 223.99415546801004)
(163.39173838319994, 148.106802618796)	(188.39135823090675, 220.66323818042082)	(294.2758703161118, 217.9725835579976)
(164.63942831574167, 144.76893683874368)	(178.79756698525006, 220.50677326537635)	(296.01614057485114, 214.44869194636988)
(165.82462252913473, 146.82565572209302)	(182.1673051599683, 215.79159740836667)	(296.8805769335147, 212.8038336379111)
(168.87014545521873, 149.45136216431547)	(180.1535796940334, 213.02560764719743)	(296.612916915066, 211.8874936515067)
(168.07087894980572, 150.56819352764234)	(179.90039039379133, 213.23720124461084)	(296.7098428790118, 168.50608445199776)
(171.91516414220206, 150.95641994203783)	(179.10105438824212, 212.67263871190187)	(295.5988578515389, 167.1691807827046)
(175.6547982828173, 149.42863906446325)	(179.93064367392088, 212.3026489538882)	(295.2327262600571, 166.7574038745105)
(195.27809017730232, 145.22636473223747)	(186.45222265666365, 211.77991953854928)	(295.01121598071285, 163.13642650226166)
(193.6500545644916, 149.97216973925345)	(190.01172389024833, 206.31899059085896)	(293.06817545312396, 162.8286651091829)
(195.2327342318892, 152.89209322695433)	(191.9664587446989, 199.96060296834003)	(290.0886216639686, 162.61025591241918)
(204.22182486800614, 152.74375527996904)	(190.67218582377325, 194.10119531415808)	(289.34869957975985, 163.14898439740188)
(203.122879906347, 147.33683042384132)	(188.48870770645334, 193.41945967646245)	(289.00967743890953, 165.77270937184704)
(201.09187536400475, 144.49946585361005)	(185.2987808767038, 194.49911547731762)	(294.0832587961736, 172.311433372394)
(214.08808060077996, 151.08954181507215)	(185.0313002233645, 199.61007123592861)	(289.47290946791526, 181.59935070957442)
(212.49579245316534, 151.00365662330887)	(183.4067406248745, 196.37236470957208)	(291.0656884803462, 180.12508285078053)
(211.17399103217474, 155.64215064123198)	(181.4666503273922, 201.45280786222736)	(294.78569368183344, 183.481517831742)
(210.00929595757646, 155.43214997674932)	(183.61287158251798, 203.92135744156496)	(295.4015783886987, 183.48209571013928)
(213.83647268581632, 159.9814126974154)	(181.09324260659082, 204.06445448888627)	(292.7422955182753, 178.20785869821168)
(214.358878413441, 154.64825483633064)	(183.09676348172377, 205.52358982358533)	(296.1565821968189, 190.14930821649267)
(218.33979693650372, 154.49537938237734)	(176.46577932753547, 206.93406570014895)	(288.2852364913426, 186.4468626410129)
(222.45312534746392, 155.857572149666)	(173.11785556662755, 197.84878391897092)	(289.49167784326346, 188.11623361827094)
(221.61989553133202, 148.1723405645796)	(175.73242534017027, 195.69565754248117)	(286.7015587049702, 191.50954005951513)
(221.68301954708352, 150.3172858426185)	(169.38237106241638, 192.63640885436098)	(280.9089260495073, 178.78707008711683)
(219.6240834936399, 147.28846091815757)	(167.29003387457124, 199.72819756571118)	(282.5538932310092, 183.27766934108678)
(217.13209585050083, 146.3679200786019)	(161.05308371244877, 196.07294616557493)	(272.6028439043511, 180.52421891635157)
(243.05319029719874, 144.3568376196465)	(160.9086378469355, 203.55475278323343)	(274.10912110099093, 181.35791548536758)
(249.8784150700542, 147.41947818008225)	(164.9759280663826, 202.0517555054409)	(278.0371640747282, 191.0258461301416)
(251.72046780984545, 146.64758333786898)	(166.01665703842758, 213.55783632610928)	(270.2720367363639, 191.81274083194984)
(251.12574447584979, 145.57011175721863)	(170.41236472787818, 210.90210249594492)	(269.4780255896581, 190.64795785444784)
(255.29453652676563, 146.20795856554884)	(170.33668580225145, 209.28374920177413)	(270.1401023839804, 185.2518076535518)
(254.3711310722972, 147.60605643562974)	(175.07044808299344, 213.38992901072464)	(258.15564025691947, 190.9193233442578)
(253.06935855545856, 147.86489989682968)	(174.48279797079795, 217.587208089589)	(258.13546650691114, 191.31943874984813)
(252.11833251936093, 147.44675763749876)	(174.0202284060424, 221.11045970441666)	(259.4733675125135, 184.24072159801403)
(254.96979348885196, 151.31431423926267)	(170.4663493269739, 223.94319570440666)	(262.4511829012746, 181.50961786235462)
(250.31621199780216, 148.78811974429948)	(168.31788076848704, 220.2517625668509)	(258.9729300102342, 180.88352428756747)
(248.6990515584692, 148.52771307875275)	(168.51608689550454, 221.756814195542)	(262.33585790721185, 179.84491176557464)
(253.7979295036692, 158.72477686578674)	(153.19502523746394, 218.32820611463853)	(263.49621604076106, 176.3484179926964)
(243.10992444908743, 154.54672250464108)	(154.4736175664715, 223.5034339767626)	(269.63793856851333, 176.29531672488696)
(241.5199224598905, 157.83124722801028)	(146.92210062053238, 218.2459259213054)	(271.4858436279879, 175.38022791798693)
(237.04329920375585, 159.7404878661577)	(149.38115291072575, 215.9878674254416)	(270.399228299531, 173.18966729769483)
(237.48355736685795, 151.58529717394123)	(146.7188131706357, 214.6448798081356)	(267.6163706993582, 169.76677812856758)
(236.56515763560134, 144.69480891962553)	(153.199096225502, 211.89173791173494)	(267.9817349608223, 173.78449881386138)
(228.2474174164309, 151.19417867832158)	(152.93431780081858, 213.47868573222985)	(266.09459657956677, 175.77062096410492)
(230.96110554764238, 147.85069587606517)	(157.54130757968204, 214.3091796662727)	(257.46419941983527, 170.90769961337645)
(224.72772845531773, 148.91700390901906)	(156.7865296990749, 211.91241397048947)	(260.4139313288459, 169.10017193372192)
(224.09004815944246, 149.77539162749446)	(158.25766831099418, 208.315659469794)	(263.41184682806886, 169.13931433969873)
(225.55472480137024, 152.27648580421106)	(158.56893564288498, 200.02274766694893)	(263.5505990441374, 168.22190001268916)
(225.3996235454839, 154.00417462297258)	(159.1914901974506, 193.89886590638417)	(262.5710084442247, 165.1290300958737)
(224.5931644864972, 156.749813400807)	(156.49698240320174, 193.5198476168979)	(264.8496578427962, 164.21142112421416)
(229.94324293920357, 160.27949675296526)	(148.19123819906167, 199.12079028952866)	(266.4622882911477, 166.47572594856223)
(229.08442292720628, 160.74930322613687)	(146.15676476490876, 195.62899969261076)	(271.7346781541689, 165.07726410217555)
(228.11740345850058, 163.53575768317074)	(150.74205183357958, 203.51990372786048)	(269.62180679529257, 160.00016252183792)
(228.56798422423327, 170.87613516600658)	(139.4158242472866, 207.2737655105447)	(269.24426267503384, 160.09446866969682)
(232.04967796542564, 171.38031575690457)	(139.14794622682393, 202.8968569445771)	(276.5257445498715, 161.07949062072595)
(234.54861703119576, 172.75291884923718)	(140.76505002270972, 202.8664757178505)	(279.8974446267939, 167.69558535582155)
(237.81802162789666, 173.720086702476)	(143.81324135049385, 202.1230405443722)	(279.89975883945266, 167.4685535759175)
(236.4495594901448, 164.59434168703126)	(142.6182170289619, 200.36668986186453)	(277.3657984565269, 173.98114559416086)
(237.96784409042937, 166.0496386132209)	(138.08355305245, 195.21730519979616)	(280.40960619036355, 169.83248781313017)
(235.87064240210145, 166.6444989251906)	(128.76456361249313, 195.92662996273032)	(282.69183364265666, 168.14542585832652)
(232.69925378943367, 162.73659487277712)	(129.23689697070904, 199.7760225595261)	(284.4517844034193, 167.34082741818247)
(236.62519787694924, 163.50340976492578)	(129.0514500099723, 200.03163164002615)	(281.95495395710657, 166.15146003731104)
(243.71371471096217, 162.65434968586464)	(133.22025023052532, 202.61884766413544)	(281.9732439165483, 165.4864920516281)
(240.1814489346317, 165.90919285682168)	(133.73620073253812, 206.4536474412095)	(287.7198578974071, 162.24269591550984)
(244.44364579437695, 164.40938317818535)	(128.63670377841697, 214.9213991214632)	(284.60147167118413, 152.8523355235958)
(245.50874884953214, 164.60768771499374)	(135.95210968147364, 215.08808114152328)	(284.1034819959091, 153.81609193828115)
(245.79314601828918, 166.90630517164908)	(134.1276432505677, 213.03214550073068)	(287.65815044234347, 148.93168298673766)
(246.38400701522875, 166.27709759467098)	(134.8621158322208, 210.3302919367726)	(273.2604842934498, 149.10757623149135)
(247.64887219060552, 163.02810662990743)	(140.76338878188517, 212.97237991556722)	(277.58264281835346, 156.98885408306933)

```
(246.035341275429, 163.99103981499036)
(248.3343706423601, 162.20346336554178)
(253.09154193892743, 161.33356995964095)
(254.4410789675561, 162.88204598575788)
(254.7221943933768, 174.22904507786677)
(245.67246470103578, 174.5719568969273)
(246.93299101356922, 168.67560914728898)
(240.2732726216488, 174.95249989377675)
(241.88397456938822, 177.42267230477324)
(241.5228896297607, 180.34922232655913)
(244.11749941641642, 183.67870959634348)
(246.2042951647768, 183.58164680988887)
(245.55852646456998, 179.34597449081565)
(245.39550016927052, 176.81422316159006)
(252.1807518243071, 176.97692564726924)
(244.64571997730678, 191.9816227741294)
(244.74138409989553, 191.3038151865025)
(233.86025723777323, 185.91064901627192)
(228.42534072616365, 180.47992359682848)
(227.95258979767692, 187.4112770667125)
(225.60856307561505, 190.01530855533105)
(219.9620808189124, 184.34711143449323)
(216.90948927977018, 188.8963856306496)
(216.51455691398263, 190.3617991610178)
(215.2178617655814, 190.68491090305616)
(208.16215180099658, 190.2817426891977)
(208.4850112786338, 184.80283624900392)
(215.92029548963933, 187.80827238012972)
(214.04809296532488, 184.31613699111216)
(215.64874480577635, 180.36405737058635)
(210.25625588843178, 179.28657255326556)
(213.50017308362305, 176.31954534518871)
(212.39638877163895, 179.14990941764472)
(138.92357183000473, 215.36116177592567)
(143.59721667377733, 223.53615322296906)
(132.74706960015538, 222.55880680714836)
(134.0205283054736, 220.70387855605708)
(133.75742884431213, 216.04914540285836)
(128.77335863108482, 222.34289271645582)
(138.213128443196, 227.8998108361992)
(140.10742099440796, 226.67019887999442)
(152.83913574386276, 225.51044930954066)
(121.54932422833502, 226.31623795733452)
(114.06239510034078, 227.8753101116863)
(111.12783794587187, 192.67542163635056)
(109.3351778923556, 194.20831235566288)
(111.16094115619876, 198.42865238320758)
(110.17086192086089, 223.52353649679003)
(115.0670170517592, 208.67106921815054)
(112.85853463766523, 212.47108569014227)
(114.16940347592413, 218.73190873447328)
(112.78941592538541, 223.10410019060794)
(117.97812226663854, 220.00826836519104)
(119.24467388842436, 223.4625722114949)
(119.15754286315797, 221.90193231184253)
(118.69905267401396, 217.83266818376399)
(126.31480687652191, 220.08621752906197)
(124.36471529528853, 212.72041418651298)
(120.25855765758422, 213.94170931433024)
(127.06290531359866, 207.59634125188842)
(124.71486952091539, 202.8763334063358)
(121.79298569928795, 201.06580458590486)
(113.5988331203668, 204.29139784916526)
(113.0063630742228, 202.5938025593856)
(115.73321116873751, 200.97376522131714)
(115.26111825459293, 199.4005790495549)
(278.1795027468564, 158.39300732264599)
(273.94639309903005, 156.10857031061585)
(273.4705903709463, 159.63474801689432)
(271.7308652785033, 159.55180907684502)
(268.54513943657315, 155.1325894216459)
(266.86069831702133, 154.30879763024518)
(261.532647444574, 159.5215887438177)
(258.9989538611589, 156.1306750300218)
(259.6888137357566, 154.26185609923664)
(263.9724475136784, 155.69113224351997)
(259.33240643708314, 146.88806700091703)
(256.52944395318343, 147.3754119236197)
(264.53306991816714, 145.8715581242713)
(267.4348873696982, 146.15416281374036)
(264.0738087349791, 147.91190003571126)
(267.92243932136813, 151.31993298535065)
(270.4606646747478, 151.25472378259593)
(270.6220537447279, 149.2540289418302)
(270.711242762453, 145.1072102487626)
(295.21050088540164, 144.22534705519706)
(290.59388761331303, 157.71385044941735)
Time = 2196 milliseconds
```

3.2 遇到过的问题

前后总共遇到了五个重大问题

1）权限问题。

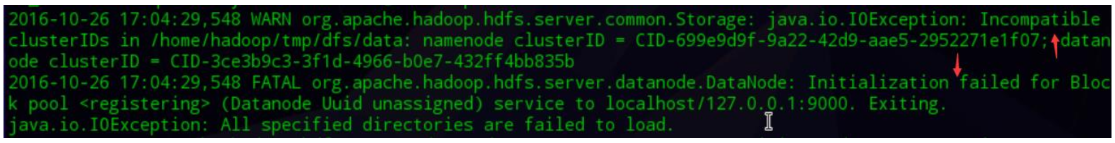
在操作的时候，遇到了“权限不够”的问题。原来我下载 hadoop，HBase 的时候使用的是我的个人账户，然而我在运行 hadoop，HBase 之前，切换到了 hadoop 账户。此时此刻需要把 hadoop，HBase 下面的所有文件所有权都改成 hadoop 账户。总之文件所有权需要统一。

2）Error: JVM terminated, exit code = 1。

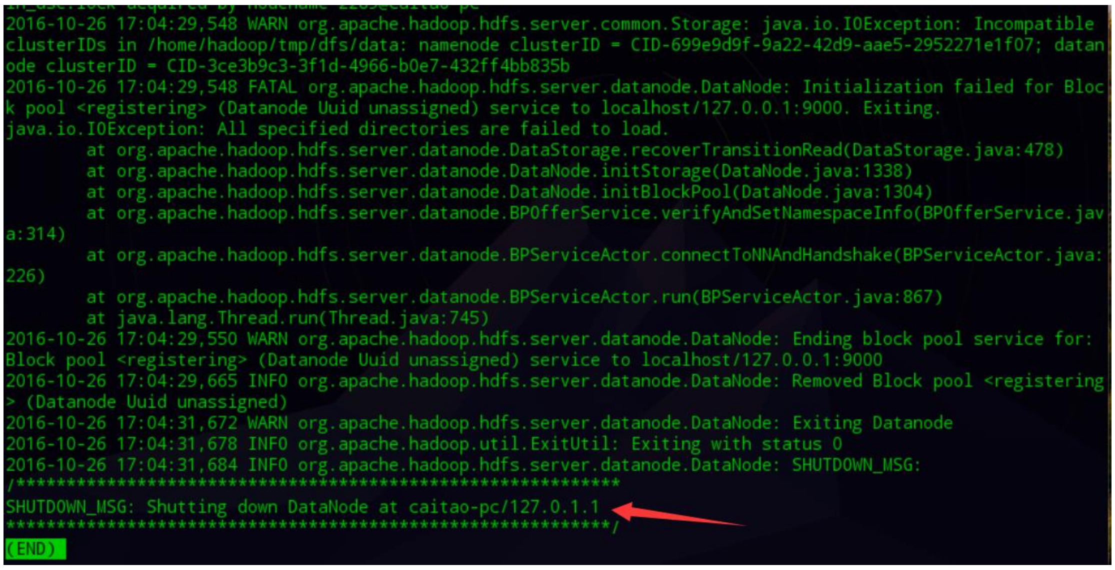
我是在虚拟机里面操作的，操作系统使用 deepin15.2，开了 2G 的内存空间。有一次当我试图打开 Eclipse 的时候，返回了一长串错误信息“Error: JVM terminated, exit code = 1……”。网上简单搜索了一下，感觉可能是因为内存不够用。打开 deepin 的资源管理器，果然此刻内存已经是 1.8G 了，很接近 2G 的上限了。我发现 hdfs, yarn, hbase 这当个东西同时打开起码耗费 1G 的内存。把这当个关掉之后，Eclipse 就可以打开了。Eclipse 也是一个吃内存的家伙，起码 0.5G。

3）scan 的时候，我设置的 SingleColumnValueFilter 过滤的与我所期望的不一样，发现过滤了不该过滤的。蛋疼死了，问题没有解决。后来发现，是由于三个小 Bug 累积叠加而成的：一，60＜500，但是”60” ＞ “500”。原来内部是按照 byte array 的字典排序的，于是在往 HBase 插入数据的时候，插入”060”，而不是”60”；二，某一处下标问题，(x, y) → [y][x] 这类问题；三，split 拆分 hilbert 值的时候，范围划分失误。

4）Hadoop 的 datanode 无法启动。一个无法理解的奇葩问题，反正解决了，直接上图。



```
2016-10-26 17:04:29,548 WARN org.apache.hadoop.hdfs.server.common.Storage: java.io.IOException: Incompatible
clusterIDs in /home/hadoop/tmp/dfs/data: namenode clusterID = CID-699e9d9f-9a22-42d9-aae5-2952271e1f07; datan
ode clusterID = CID-3ce3b9c3-3f1d-4966-b0e7-432ff4bb835b
2016-10-26 17:04:29,548 FATAL org.apache.hadoop.hdfs.server.datanode.DataNode: Initialization failed for Bloc
k pool <registering> (Datanode Uuid unassigned) service to localhost/127.0.0.1:9000. Exiting.
java.io.IOException: All specified directories are failed to load.
```



```
2016-10-26 17:04:29,548 WARN org.apache.hadoop.hdfs.server.common.Storage: java.io.IOException: Incompatible
clusterIDs in /home/hadoop/tmp/dfs/data: namenode clusterID = CID-699e9d9f-9a22-42d9-aae5-2952271e1f07; datan
ode clusterID = CID-3ce3b9c3-3f1d-4966-b0e7-432ff4bb835b
2016-10-26 17:04:29,548 FATAL org.apache.hadoop.hdfs.server.datanode.DataNode: Initialization failed for Bloc
k pool <registering> (Datanode Uuid unassigned) service to localhost/127.0.0.1:9000. Exiting.
java.io.IOException: All specified directories are failed to load.
        at org.apache.hadoop.hdfs.server.datanode.DataStorage.recoverTransitionRead(DataStorage.java:478)
        at org.apache.hadoop.hdfs.server.datanode.DataNode.initStorage(DataNode.java:1338)
        at org.apache.hadoop.hdfs.server.datanode.DataNode.initBlockPool(DataNode.java:1304)
        at org.apache.hadoop.hdfs.server.datanode.BPOfferService.verifyAndSetNamespaceInfo(BPOfferService.jav
a:314)
        at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.connectToNNAndHandshake(BPServiceActor.java:
226)
        at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.run(BPServiceActor.java:867)
        at java.lang.Thread.run(Thread.java:745)
2016-10-26 17:04:29,550 WARN org.apache.hadoop.hdfs.server.datanode.DataNode: Ending block pool service for:
Block pool <registering> (Datanode Uuid unassigned) service to localhost/127.0.0.1:9000
2016-10-26 17:04:29,665 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Removed Block pool <registering
> (Datanode Uuid unassigned)
2016-10-26 17:04:31,672 WARN org.apache.hadoop.hdfs.server.datanode.DataNode: Exiting Datanode
2016-10-26 17:04:31,678 INFO org.apache.hadoop.util.ExitUtil: Exiting with status 0
2016-10-26 17:04:31,684 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down DataNode at caitao-pc/127.0.1.1
************************************************************/
(END)
```

解决方案参考了[13]，大概原因是 namenode 和 datanode 的 clusterID 不一样。解决方案是把 datanode 的 clusterID 改成和 namenode 的一样。

5）10 月 16 日采集实验数据的时候，虚拟机突然变得不稳定，时常死机。这个我无能为力，只有碰运气，希望不要死机。

4 总结

4.1 总结与体会

在这次软件综合实习中，我动手能力和理论水平都得到了很大的提升。

动手能力主要体现在 Linux 环境下的 Java 编程。因为 HBase 以及 Hadoop 都是运行在 Linux 环境下的，我在 Linux 环境下的安装和配置水平的提升很大。在解决了几次困难之后，对于 Linux 再也没有什么“畏惧”之感了。

理论水平主要体现于对于 HBase，这一种流行的 NoSQL 数据库的理解。我掌握了一种面向列存储的数据库的基础知识以及设计原则。现在市面上有不少非关系型数据库，作为计算机专业的学生，有必要对这些非关系型数据库做了解甚至掌握。

4.2 系统存在的问题

系统没有啥问题。除了 oracle 虚拟机有时候不稳定。

4.3 将来的工作

1. 现在是在单台机器上模拟集群，伪分布式模式。将来可以在真正的集群上进行实验
2. 现在只能对点进行存储和检索，将来可以对线和多边形进行处理

4.4 参考文献

[1] http://hbase.apache.org/

[2] https://en.wikipedia.org/wiki/Geographic_data_and_information

[3] https://www.shiyanlou.com/courses/35

[4] https://www.shiyanlou.com/courses/37

[5] https://en.wikipedia.org/wiki/Hilbert_curve

[6] http://blog.jobbole.com/81106/, 《超酷算法：用四叉树和希尔伯特曲线做空间索引》

[7] http://blog.jobbole.com/80633/, 《GeoHash 核心原理解析》

[8] http://blog.csdn.net/firecoder/article/details/7178928, 《Hilbert 曲线》

[9]张榆，马友忠，孟小峰，一种基于 HBase 的高效空间关键字查询策略，Journal of Chinese Computer Systems，2012 年 10 月第 10 期，Vol.33 No.10

[10]丁琛，基于 HBase 的空间数据分布式存储和并行查询算法研究_丁琛，南京师范大学，硕士学位论文

[11] L. Wang, B. Chen, Y. Liu, Distributed Storage and Index of Vector Spatial Data Based on HBase, 2013 21st International Conference on Geoinformatics

[12] K. Zheng, Y. Fu, Research on Vector Spatial Data Storage Schema Based on Hadoop Platform, International Journal of Database Theory and Application, 2013, Vol.6 No.5

[13] http://www.linuxidc.com/Linux/2014-11/108822.htm，《Hadoop 启动节点 Datanode 失败解决》