HBase 中加盐（Salting）之后的表如何读取：Spark 篇

在《HBase 中加盐（Salting）之后的表如何读取：协处理器篇》
文章中介绍了使用协处理器来查询加盐之后的表，本文将介绍第二种方法来实现相同的功能。

我们知道，HBase 为我们提供了 hbase-mapreduce 工程包含了读取 HBase 表的 InputFormat、OutputFormat 等类。这个工程的描述如下：
This module contains implementations of InputFormat, OutputFormat, Mapper, Reducer, etc which are needed for running MR jobs on tables, WALs, HFiles and other HBase specific constructs. It also contains a bunch of tools: RowCounter, ImportTsv, Import, Export, CompactionTool, ExportSnapshot, WALPlayer, etc.
我们也知道，虽然上面描述的是 MR jobs，但是 Spark 也是可以使用这些 InputFormat、OutputFormat 来读写 HBase 表的，如下：

```
val sparkSession = SparkSession.builder
  .appName("HBase")
  .getOrCreate()

val conf = HBaseConfiguration.create()
conf.set("hbase.zookeeper.quorum", "https://www.iteblog.com:2181")
conf.set(TableInputFormat.INPUT_TABLE, "iteblog")

val HBaseRdd = sparkSession.sparkContext.newAPIHadoopRDD(conf, classOf[TableInputFormat],
  classOf[ImmutableBytesWritable],
  classOf[Result])

println(HBaseRdd.count())
```

上面程序使用 TableInputFormat 计算了 iteblog 表的总行数。如果我们想查询某个 UID 的所有历史记录如何实现呢？如果你查看 TableInputFormat 代码，你会发现其包含了很大参数设置：

```
hbase.mapreduce.inputtable
hbase.mapreduce.splittable
hbase.mapreduce.scan
hbase.mapreduce.scan.row.start
hbase.mapreduce.scan.row.stop
hbase.mapreduce.scan.column.family
hbase.mapreduce.scan.columns
```

hbase.mapreduce.scan.timestamp
hbase.mapreduce.scan.timerange.start
hbase.mapreduce.scan.timerange.end
hbase.mapreduce.scan.maxversions
hbase.mapreduce.scan.cacheblocks
hbase.mapreduce.scan.cachedrows
hbase.mapreduce.scan.batchsize
hbase.mapreduce.inputtable.shufflemaps

其中 hbase.mapreduce.inputtable 就是需要查询的表，也就是上面 Spark 程序里面的 TableInputFormat.INPUT_TABLE。而 hbase.mapreduce.scan.row.start 和 hbase.mapreduce.scan.row.stop 分别对应的是需要查询的起止 Rowkey，所以我们可以利用这个信息来实现某个范围的数据查询。但是要注意的是，iteblog 这张表是加盐了，所以我们需要在 UID 之前加上一些前缀，否则是查询不到数据的。不过 TableInputFormat 并不能实现这个功能。那如何处理呢？答案是重写 TableInputFormat 的 getSplits 方法。

从名字也可以看出 getSplits 是计算有多少个 Splits。在 HBase 中，一个 Region 对应一个 Split，对应于 TableSplit 实现类。TableSplit 的构造是需要传入 startRow 和 endRow。startRow 和 endRow 对应的就是上面 hbase.mapreduce.scan.row.start 和 hbase.mapreduce.scan.row.stop 参数传进来的值，所以如果我们需要处理加盐表，就需要在这里实现。

另一方面，我们可以通过 RegionLocator 的 getStartEndKeys() 拿到某张表所有 Region 的 StartKeys 和 EndKeys 的。然后将拿到的 StartKey 和用户传进来的 hbase.mapreduce.scan.row.start 和 hbase.mapreduce.scan.row.stop 值进行拼接即可实现我们要的需求。根据这个思路，我们的代码就可以按照如下实现：

```
package com.iteblog.data.spark;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.google.common.base.Strings;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.RegionLocator;
import org.apache.hadoop.hbase.mapreduce.TableInputFormat;
import org.apache.hadoop.hbase.mapreduce.TableSplit;
import org.apache.hadoop.hbase.util.Bytes;
```

```
import org.apache.hadoop.hbase.util.Pair;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;

public class SaltRangeTableInputFormat extends TableInputFormat {

  @Override
  public List<InputSplit> getSplits(JobContext context) throws IOException {
    Configuration conf = context.getConfiguration();

    String tableName = conf.get(TableInputFormat.INPUT_TABLE);
    if (Strings.isNullOrEmpty(tableName)) {
      throw new IOException("tableName must be provided.");
    }

    Connection connection = ConnectionFactory.createConnection(conf);
    val table = TableName.valueOf(tableName)
    RegionLocator regionLocator = connection.getRegionLocator(table);


    String scanStart = conf.get(TableInputFormat.SCAN_ROW_START);
    String scanStop = conf.get(TableInputFormat.SCAN_ROW_STOP);

    Pair<byte[][], byte[][]> keys = regionLocator.getStartEndKeys();
    if (keys == null || keys.getFirst() == null || keys.getFirst().length == 0) {
      throw new RuntimeException("At least one region is expected");
    }
    List<InputSplit> splits = new ArrayList<>(keys.getFirst().length);
    for (int i = 0; i < keys.getFirst().length; i++) {
      String regionLocation = getTableRegionLocation(regionLocator, keys.getFirst()[i]);
      String regionSalt = null;
      if (keys.getFirst()[i].length > 0) {
        regionSalt = Bytes.toString(keys.getFirst()[i]).split("-")[0];
      }

      byte[] startRowKey = Bytes.toBytes(regionSalt + "-" + scanStart);
      byte[] endRowKey = Bytes.toBytes(regionSalt + "-" + scanStop);

      InputSplit split = new TableSplit(TableName.valueOf(tableName),
          startRowKey, endRowKey, regionLocation);
      splits.add(split);
    }
    return splits;
  }

  private String getTableRegionLocation(RegionLocator regionLocator,
```

```
                                    byte[] rowKey) throws IOException {
      return regionLocator.getRegionLocation(rowKey).getHostname();
    }
}
```

然后我们同样查询 UID = 1000 的用户所有历史记录，那么我们的程序可以如下实现：

```
package com.iteblog.data.spark

import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.util.Bytes
import org.apache.spark.sql.SparkSession

import scala.collection.JavaConversions._

object Spark {
  def main(args: Array[String]): Unit = {
    val sparkSession = SparkSession.builder
      .appName("HBase")
      .getOrCreate()

    val conf = HBaseConfiguration.create()
    conf.set("hbase.zookeeper.quorum", "https://www.iteblog.com:2181")
    conf.set(TableInputFormat.INPUT_TABLE, "iteblog")
    conf.set(TableInputFormat.SCAN_ROW_START, "1000")
    conf.set(TableInputFormat.SCAN_ROW_STOP, "1001")

    val HBaseRdd = sparkSession.sparkContext.newAPIHadoopRDD(conf, classOf[SaltRangeTabl
eInputFormat],
      classOf[ImmutableBytesWritable],
      classOf[Result])

    HBaseRdd.foreach { case (_, result) =>
      val rowKey = Bytes.toString(result.getRow)
      val cell = result.listCells()
      cell.foreach { item =>
        val family = Bytes.toString(item.getFamilyArray, item.getFamilyOffset, item.getFamilyLengt
h)
        val qualifier = Bytes.toString(item.getQualifierArray,
          item.getQualifierOffset, item.getQualifierLength)
```

```
      val value = Bytes.toString(item.getValueArray, item.getValueOffset, item.getValueLength)
      println(rowKey + " ₩t " + "column=" + family + ":" + qualifier + ", " +
        "timestamp=" + item.getTimestamp + ", value=" + value)
    }
  }
 }
}
```

我们编译打包上面的程序，然后使用下面命令运行上述程序：

```
bin/spark-submit --class com.iteblog.data.spark.Spark
    --master yarn
    --deploy-mode cluster
    --driver-memory 2g
    --executor-memory 2g ~/hbase-1.0-SNAPSHOT.jar
```

得到的结果如下：

```
A-1000-1550572395399   column=f:age, timestamp=1549091990253, value=54
A-1000-1550572395399   column=f:uuid, timestamp=1549091990253, value=e9b10a9f-1218-4
3fd-bd01
A-1000-1550572413799   column=f:age, timestamp=1549092008575, value=4
A-1000-1550572413799   column=f:uuid, timestamp=1549092008575, value=181aa91e-5f1d-4
54c-959c
A-1000-1550572414761   column=f:age, timestamp=1549092009531, value=33
A-1000-1550572414761   column=f:uuid, timestamp=1549092009531, value=19aad8d3-621a-4
73c-8f9f
B-1000-1550572388491   column=f:age, timestamp=1549091983276, value=1
B-1000-1550572388491   column=f:uuid, timestamp=1549091983276, value=cf720efe-2ad2-48
d6-81b8
B-1000-1550572392922   column=f:age, timestamp=1549091987701, value=7
B-1000-1550572392922   column=f:uuid, timestamp=1549091987701, value=8a047118-e130-4
8cb-adfe
.....
```

和前面文章使用 HBase Shell 输出结果一致。

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）