pitney bowes

Spectrum™ Routing for Big Data

Version 3.1

Spectrum™ Routing for Big Data Install Guide: MapR

Table of Contents

1 - Welcome

In this section

Introducing Spectrum™ Routing for Big Data

Spectrum™ Routing for Big Data is a toolkit that can be used for processing enterprise data for large-scale spatial analysis. Billions of records from a single file can be processed in parallel using Hive's cluster processing framework, yielding results faster than ever. Unlike the traditional processing techniques that took weeks to process data, using this product, data processing can now be done in a just a few hours.

System Requirements and Dependencies

Spectrum™ Routing for Big Data is collection of jar files that can be deployed to your Hadoop system. This product is verified on the following Hadoop distribution:

- MapR 6.0 and above, with MapR Expansion Pack (MEP) 5.0.0

To use these jar files, you must be familiar with configuring Hadoop in MapR and developing applications for distributed processing. For more information, refer to **MapR** documentation.

To use the product, the following must be installed on your system:

for Hive:

- Hive version 1.2.1 or above

for Hive Client

- Beeline, for example

for Spark and Zeppelin Notebook:

- Java JDK version 1.8 or above
- Hadoop version 2.6.0 or above
- Spark version 2.0 or above

2 - Data Installation

In this section

Before You Begin

Before you can use the Routing function, you must install routing data following the instructions in the Release Notes that accompany the data.

A data resource file (dbList.json) that contains the datasets and database configurations used is also required. A sample dbList.json file is provided in the distribution at `spectrum-bigdata-routing-version.zip/resources/config`:

```
{
 "defaultDatabase": "US",
 "datasets": [{
   "id": "US",
   "paths": ["hdfs:///pb/routing/data/US_Driving/central",
"hdfs:///pb/routing/data/US_Driving/midwest",
"hdfs:///pb/routing/data/US_Driving/northeast",
"hdfs:///pb/routing/data/US_Driving/pacific",
"hdfs:///pb/routing/data/US_Driving/south"]
  }
 ],
 "databases": [{
   "name": "US",
   "datasets": ["US"]
  }
 ]
}
```

Note: If you are using NSF mounts on MapR or S3, change the paths in the dbList.json accordingly.

Installing the SDK

For the purposes of this guide, we will:

- use a user called pbuser
- install everything into `/pb`
- use installed routing datasets

Perform the following steps from a node in your cluster, such as the master node.

1. Create the install directory and give ownership to pbuser.

```
sudo mkdir /pb
sudo chown pbuser:pbuser /pb
```

2. Add the routing distribution zip to the node at a temporary location, for example:

```
/pb/temp/spectrum-bigdata-routing-version.zip
```

3. Extract the routing distribution.

```
mkdir /pb/routing
mkdir /pb/routing/sdk
unzip /pb/temp/spectrum-bigdata-routing-version.zip -d /pb/routing/sdk
```

4. You will now need to distribute or install the reference data. The recommended method is via NFS (see **Installing Reference Data on a MapR File System** on page 9). Alternatively, you can distribute the reference data remotely, although slower performance could result (see **Distributing Reference Data Using HDFS** on page 9 or **Distributing Reference Data Using S3** on page 10).

Distributing Reference Data

Installing Reference Data on a MapR File System

Now that the SDK is installed, the reference data must be distributed around the cluster on the MapR file system.

Important: This approach requires MapR Direct Access NFS™. For more information, see the following MapR documentation:

- **MapR Direct Access NFS™** (white paper)
- **Direct Access NFS™** (MapR Installation Guide)

For the purposes of this guide, we will:

- continue using pbuser
- install the reference data into `/mapr/<cluster-name>/pb/routing/data`

1. Create an install directory on the MapR file system and give ownership to pbuser.

```
sudo hadoop fs -mkdir /mapr/<cluster-name>/pb
sudo hadoop fs -chown pbuser:pbuser /mapr/<cluster-name>/pb
```

2. Upload the reference data onto the MapR file system.

```
hadoop fs -mkdir /mapr/<cluster-name>/pb/routing
hadoop fs -copyFromLocal /pb/routing/data
/mapr/<cluster-name>/pb/routing/data
```

Distributing Reference Data Using HDFS

Now that the SDK is installed and the routing reference data is configured the reference data must be distributed around the cluster.

For the purposes of this guide, we will:

- continue using pbuser
- install the reference data into `hdfs:///pb/routing/data`

1. Create an install directory on hdfs and give ownership to pbuser.

```
sudo -u hdfs hadoop fs -mkdir hdfs:///pb
sudo -u hdfs hadoop fs -chown pbuser:pbuser hdfs:///pb
```

2. Upload the reference data into HDFS.

```
hadoop fs -mkdir hdfs:///pb/routing
hadoop fs -copyFromLocal /pb/routing/data hdfs:///pb/routing/data
```

3. When the data node performs routing tasks, the node will download the reference data from HDFS and onto the local file system. This means a local directory needs to be set up on all data nodes. Perform the following commands on all data nodes and HiverServer nodes.

```
sudo mkdir /pb/downloads
sudo chown pbuser:hadoop /pb/downloads
sudo chmod 775 /pb/downloads
```

Distributing Reference Data Using S3

Now that the SDK is installed and the routing reference data is configured the reference data must be distributed around the cluster. Your cluster must be set up for S3 access.

Use the s3a URI scheme on MapR distributions.

For the purposes of this guide, we will install the reference data into `s3a://<your-bucket>/pb/routing/data`.

1. Upload the reference data into S3.

```
hadoop fs -mkdir -p s3a://<your-bucket>/pb/routing
hadoop fs -copyFromLocal /pb/routing/data
s3a://<your-bucket>/pb/routing/data
```

2. When the data node performs routing tasks, the node will download the reference data from S3 and onto the local file system. This means a local directory needs to be set up on all data nodes. Perform the following commands on all data nodes and HiverServer nodes.

```
sudo mkdir /pb/downloads
sudo chown pbuser:hadoop /pb/downloads
sudo chmod 775 /pb/downloads
```

3 - Routing Setup

In this section

Hive Setup

To set up routing for Hive, perform the following steps from the same node used in **Installing the SDK** on page 8:

1. Copy the Hive routing jar to the HiveServer node:

```
/pb/routing/sdk/hive/lib/spectrum-bigdata-routing-hive-version.jar
```

2. Restart all Hive services.
3. Launch Beeline, or some other Hive client, for the remaining steps:

```
beeline -u jdbc:hive2://localhost:10000/default -n pbuser
```

4. Create the Routing functions and set the Hive variables. Add the `temporary` keyword after `create` if you want a temporary function (this step would need to be redone for every new Hive session). For more information, see **Hive Variables** on page 13.

   Example

```
create function PointToPointRoute as 'com.pb.bigdata.spatial.routing.hive.PointToPointRoute';
create function IsoChrone as 'com.pb.bigdata.spatial.routing.hive.IsoChrone';
create function IsoDistance as 'com.pb.bigdata.spatial.routing.hive.IsoDistance';
set pb.routing.config.location=/mapr/<cluster-name>/pb/routing/config/dbList.json;
set pb.routing.engine.timeout=10000;
set pb.routing.allowFallback=true;
```

5. Set `hive.aux.jars.path` in hive-site.xml and (for Hive v2.1 and earlier only) `HIVE_AUX_JARS_PATH` in hive-env.sh using full paths to the jar files (not to folders) for only the nodes that are running HiveServer2 or Hive metastore (that is, the master node or nodes).

   - `/opt/mapr/hive/hive-version/conf/hive-site.xml`

     Qualify the `hive.aux.jars.path` with the `file://uri` prefix and separate multiple paths with a comma.

```
<property>
  <name>hive.aux.jars.path</name>
  <value> file:///pb/routing/sdk/hive/lib/spectrum-bigdata-routing-hive-version.jar,

          file:///pb/geocoding/sdk/hive/lib/spectrum-bigdata-geocoding-hive-version.jar

  </value>
</property>
```

   - `/opt/mapr/hive/hive-version-2.1-or-earlier/conf/hive-env.sh`

Export the environment variable and separate multiple paths with a colon (`:`).

```
export HIVE_AUX_JARS_PATH=/pb/routing/sdk/hive/lib/
spectrum-bigdata-routing-hive-version.jar:/pb/geocoding/sdk/hive/lib/
spectrum-bigdata-geocoding-hive-version.jar
```

6. Test the Routing function.

```
SELECT IsoDistance(-77.088217, 38.937072, 3, map('distanceUnit',
'km'));
```

Note:

- Some types of queries will cause Hive to evaluate UDFs in the HiveServer2 process space instead of on a data node. The Routing UDFs in particular use a significant amount of memory and can shut down the Hive server due to memory constraints. To process these queries, we recommend increasing the amount of memory available to the HiveServer2 process (for example, by setting HADOOP_HEAPSIZE in hive-env.sh).

This query should return a polygon geometry comprising all the points that lie at the specified distance from the starting point.

Hive Variables

The Routing UDFs use Hive variables to set various properties, which take precedence over the system properties. They can also be set as part of an option in a UDF query, where they will take precedence over the Hive variables as well as the system properties.

Variable	Required	Example
`pb.routing.config.location` Location of dbList.json containing the routing configuration.	Yes	/mapr/<*cluster-name*>/pb/routing/config/dbList.json
`pb.routing.engine.timeout` Specifies the duration (in seconds) after which a GRA engine shuts down if it does not receive any request. The default value of this property is -1, which means that the GRA engines will never time out.	No	300

Variable	Required	Example
`pb.routing.routeTimeout` Specifies the amount of time, in milliseconds, allowed for a point-to-point routing job to be processed and completed. If the job is not processed or completed within the specified duration, the SQL returns null to the console and exception to logs for each record processed. This property can be used for the PointToPointRoute UDF.	No	10000
`pb.routing.isoTimeout` Specifies the amount of time (in milliseconds) allowed for a boundary routing job to be processed and completed. If the job is not processed or completed within the specified duration, the SQL returns null to the console and exception to logs for each record processed. This property can be used for the IsoDistance and IsoChrone UDFs	No	17000
`pb.routing.allowFallback` This Boolean property controls whether the engine can use major roads in case other roads cannot be used. If set to true, the engine can use major roads.	No	false
`pb.routing.error.limit` The number of routing errors to allow before failing a task for "too many errors". This prevents a task (and thus the job) from continuing in the case of a likely configuration error. The default value is 10.	No	1
`pb.routing.error.limit.disabled` Disables the error limit. All errors will be logged but will not cause the task or job to fail.	No	true

Setting Variables

Since these are Hive configuration variables, you can set them permanently by editing the hiveserver2-site.xml file in your cluster.

```
pb.routing.config.location=/mapr/<cluster-name>/pb/routing/config/dbList.json;
pb.routing.engine.timeout=10000;
```

Alternatively you can set them temporarily in each Hive session that you open.

For example, if you wish to perform routing in Beeline you can set the variables in the following way:

```
set
pb.routing.config.location=/mapr/<cluster-name>/pb/routing/config/dbList.json;
set pb.routing.engine.timeout=10000;
set pb.routing.allowFallback=true;
```

Variables set in the local session will override any hive-site variables. This means you can have default values set in the hiveserver2-site file and override them in Beeline when necessary.

System Variables (Deprecated)

Hive variables used to exist in the system namespace. This is deprecated as of version 3.1 and no longer the recommended method.

Copyright

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com