Spectrum™ Geocoding for Big Data

Version 4.0

Spectrum™ Geocoding for Big Data Install Guide: Hortonworks

Table of Contents

1 - Welcome

What is Spectrum™ Geocoding for Big Data? 4
System Requirements and Dependencies 5

2 - Data Installation

Before You Begin 7
Installing the SDK and Configuring Reference Data 7
Distributing Reference Data 9

3 - Geocoding Setup

Hive Setup 13
Spark Setup 18

4 - Advanced Setup Options

Geocode Preferences 21
Geocode-Level Errors 22
Download Permissions 23

1 - Welcome

In this section

What is Spectrum™ Geocoding for Big Data? 4
System Requirements and Dependencies 5

What is Spectrum™ Geocoding for Big Data?

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

This guides covers installing the Spectrum™ Geocoding for Big Data SDK, configuring and distributing the reference data in your cluster, and setting up geocoding for the Hive and Spark frameworks.

System Requirements and Dependencies

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

- Hortonworks 3.x

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

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

for Hive:

- Hive version 1.2.1 or above
- Hive in Spectrum™ Geocoding for Big Data is not supported on Cloudera 5.12

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 7
Installing the SDK and Configuring Reference Data 7
Distributing Reference Data 9

Before You Begin

Before you can geocode you must install geocoding data. Then there are specific steps depending on the services you plan to use, Hive or Spark.

Installing the SDK and Configuring Reference Data

To use geocoding, the Hadoop cluster must have the geocoding reference data and geocoding resources accessible from each master and data node at the file-system level.

For the purposes of this guide, we will:

- use a user called pbuser
- install everything into `/pb`
- use two geocoding datasets: US Master Location Data (KLD*mmyyyy*.spd) and US NAVTEQ Streets (KNT*mmyyyy*.spd)

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 geocoding distribution zip to the node at a temporary location, for example:

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

3. Add the geocoding reference datasets to the cluster at the temporary location in a new data folder:

   ```
   /pb/temp/data/KLDmmyyyy.spd
   /pb/temp/data/KNTmmyyyy.spd
   ```

4. Extract the geocoding distribution.

   ```
   mkdir /pb/geocoding
   mkdir /pb/geocoding/software
   unzip /pb/temp/spectrum-bigdata-geocoding-version.zip -d /pb/geocoding/software
   ```

5. Navigate to the Geocoding CLI.

```
cd /pb/geocoding/software/cli
```

6. Using the Geocoding CLI, extract the geocoding reference data. This will extract any datasets that are a zip or spd file.

```
bash cli.sh extract --s /pb/temp/data --d /pb/geocoding/data
```

7. Optionally, if you have a dataset that is a directory and already extracted, then move the directory into the new data folder. For example, if there was a DPV dataset then you would run the command below.

```
mv /pb/temp/data/DPV /pb/geocoding/data
```

8. Using the Geocoding CLI tool, configure the geocoding reference data.

```
bash cli.sh configure --s /pb/geocoding/data --d
/pb/geocoding/software/resources/config
```

   Note: This step generates the *JsonDataConfig.json* file located in `pb/geocoding/software/resources/config`. This file contains configuration information for your geocoding datasets and needs to be regenerated for every software and data release.

9. You will now need to distribute the reference data. A remote method via HDFS or S3 is recommended (see **Distributing Reference Data Using HDFS** on page 9 or **Distributing Reference Data Using S3** on page 10). If you wish to manually distribute the reference, see **Installing Reference Data Manually on a Local File System** on page 11.

Distributing Reference Data

Distributing Reference Data Using HDFS

Now that the SDK is installed and the geocoding 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/geocoding/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/geocoding
hadoop fs -copyFromLocal /pb/geocoding/data hdfs:///pb/geocoding/data
```

3. Configure an HDFS root.

```
bash cli.sh setting --t "ROOTS" --n "root1" --v
"hdfs:///pb/geocoding/data" --d /pb/geocoding/software/resources/config
```

4. Upload the Resources directory into HDFS.

```
hadoop fs -copyFromLocal /pb/geocoding/software/resources
hdfs:///pb/geocoding/software/resources
```

5. When the data node performs geocoding 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 geocoding 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 Hortonworks distributions.

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

1. Upload the reference data into S3.

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

2. Configure an S3 root.

   ```
   bash cli.sh setting --t "ROOTS" --n "root1" --v
   "s3a://<your-bucket>/pb/geocoding/data" --d
   /pb/geocoding/software/resources/config
   ```

3. Upload the Resources directory into S3.

   ```
   hadoop fs -copyFromLocal /pb/geocoding/software/resources
   s3a://<your-bucket>/pb/geocoding/software/resources
   ```

4. When the data node performs geocoding 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
   ```

Installing Reference Data Manually on a Local File System

After the SDK is installed and the geocoding reference data is configured, the reference data must be distributed around the cluster. If you do not wish to use HDFS or S3 to do this, you can manually distribute the data instead.

1. Create the install directories on every data node and HiverServer2 node.

   ```
   sudo mkdir -p /pb/geocoding/data/
   sudo mkdir -p /pb/geocoding/software/resources/
   sudo chown -R pbuser:pbuser /pb
   ```

2. Copy the data directory to the same path on every data node and HiveServer2 node.

   ```
   /pb/geocoding/data/
   ```

3. Copy the resources directory to the same path on every data node and HiveServer2 node. This folder contains the configuration folder and the native libraries.

   ```
   /pb/geocoding/software/resources/
   ```

   Note: While it is recommended you use HDFS or S3 for the configuration directories and preference files, it is not a requirement. You may use local file system paths but you will have to make sure the file or directory is present and current on every node in the cluster. If you do this then you do not need to set `download.location`.

3 - Geocoding Setup

In this section

Hive Setup 13
Spark Setup 18

Hive Setup

To set up geocoding for Hive, perform the following steps from the same node used in **Installing the SDK and Configuring Reference Data** on page 7:

1. On the HiveServer2 node, create the Hive auxlib folder if one does not already exist.

```
sudo mkdir /usr/hdp/current/hive-server2/auxlib/
```

2. Copy the Hive geocoding jar to the folder on the HiveServer2 node created in step 1:

```
sudo cp
/pb/geocoding/software/hive/lib/spectrum-bigdata-geocoding-hive-version.jar
 /usr/hdp/current/hive-server2/auxlib/
```

3. Restart all Hive services.
4. Requested output fields can be set as a comma-separated list of fields as a Hive variable. For more information, see **Hive Output Fields** on page 17.
5. Launch Beeline, or some other Hive client, for the remaining steps:

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

6. Create the Geocode function 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). You may choose to substitute the `pb.geocoding.output.fields` variable with the `pb.geocoding.preferences.filepath` variable that points to a file. For more information, see **Hive Variables** on page 14.

```
create function Geocode as 'com.pb.bigdata.geocoding.hive.Geocode';
create function ReverseGeocode as 'com.pb.bigdata.geocoding.hive.ReverseGeocode';
create function GeocodeByKey as 'com.pb.bigdata.geocoding.hive.GeocodeByKey';
set pb.geocoding.resources.location=hdfs:///pb/geocoding/software/resources/;
set pb.geocoding.output.fields=X,Y,formattedStreetAddress,formattedLocationAddress;
set
pb.geocoding.preferences.filepath=hdfs:///pb/geocoding/software/resources/config/geocodePreferences.xml;
set pb.download.location=/pb/downloads;
set pb.download.group=pbdownloads;
```

7. Test the geocode with a single address. If you do not have USA configured then choose a country that you have a dataset for.

```
select Geocode("600 Maryland Ave SW, Washington, DC 20002", "USA");
```

Note:

- The first time you run the job may take a while if the reference data has to be downloaded remotely from HDFS or S3. It may also time out when using a large number of datasets

  that are stored in remote locations such as HDFS or S3. If you are using Hive with the MapReduce engine, you can adjust the value of the `mapreduce.task.timeout` property.
- 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).

8. The response should be a single struct response. If everything is configured properly then the error field should be null.

```
{"x" : "-77.019959", "y" : "38.886803", "formattedstreetaddress" :
"600 MARYLAND AVE SW", "formattedlocationaddress" : "WASHINGTON, DC
  20024-2520", "precisioncode" : "S5HPNTSC-A", "error" : null}
```

Hive Variables

The Geocoding UDFs use Hive variables (some of which are required) 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	Examples
`pb.geocoding.resources.location` Location of the geocoding resources directory which contains the configurations and libraries needed for geocoding. This variable must be set.	hdfs:///pb/geocoding/software/resources/
`pb.geocoding.preferences.filepath` File path of the geocoding preferences file. This optional file can be edited by advanced users to change the behavior of the geocoder.	hdfs:///pb/geocoding/software/resources/config/geocodePreferences.xml
`pb.geocoding.output.fields` Comma-separated list of fields requested from the geocoder. You must either set this variable or set the preferences in the UDF query. For more information about output fields see **Hive Output Fields**.	X,Y,formattedStreetAddress,formattedLocationAddress

Variable	Examples
`pb.download.location` **Note:** Use this variable only if reference data was distributed remotely via HDFS or S3. Location of the directory where reference data will be downloaded to. This path must exist on every data node and the HiveServer2 node.	/pb/downloads
`pb.download.group` **Note:** Use this variable only if reference data was distributed remotely via HDFS or S3. This is an optional property and only specific to POSIX-compliant platforms like Linux. It specifies the operating system group which should be applied to the downloaded data on a local file system, so that each Hadoop service can update the data when required. This group should be present on all nodes in the cluster and the operating system user executing the Hadoop service should be a part of this group. For more information, see **Download Permissions** on page 23.	pbdownloads
`pb.geocoding.error.limit` The number of geocoding 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.	1
`pb.geocoding.error.limit.disabled` Disables the error limit. All errors will be logged but will not cause the task or job to fail.	true

Setting Variables

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

Example (using HDFS)

```
pb.geocoding.resources.location=hdfs:///pb/geocoding/software/resources/
pb.geocoding.output.fields=X,Y,formattedStreetAddress,formattedLocationAddress
pb.geocoding.preferences.filepath=hdfs:///pb/geocoding/software/resources/config/geocodePreferences.xml
pb.download.location=/pb/downloads
pb.download.group=pbdownloads
pb.geocoding.error.limit=1
```

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

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

Example (using HDFS)

```
set pb.geocoding.resources.location=hdfs:///pb/geocoding/software/resources/;
set pb.geocoding.output.fields=X,Y,formattedStreetAddress,formattedLocationAddress;
set
pb.geocoding.preferences.filepath=hdfs:///pb/geocoding/software/resources/config/geocodePreferences.xml;
set pb.download.location=/pb/downloads;
set pb.download.group=pbdownloads;
set pb.geocoding.error.limit=1;
```

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.

Hive Output Fields

The result of a Geocode UDF or Reverse Geocode UDF is a struct that will look similar to the example below.

```
    {
        "x":"-73.700257",
        "y":"42.678161",
        "formattedstreetaddress":"350 JORDAN RD",
        "formattedlocationaddress":"TROY, NY 12180-8352",
        "error": null
    }
```

To see examples of queries using this struct see Geocode UDF or Reverse Geocode UDF. To change the fields present in the struct you must use either the `pb.geocoding.output.fields` variable or set the preferences in the UDF query.

> **Note:** For information on custom output fields per country, see the appendix in the ***Global Geocoding SDK Developer Guide***, which is also available on the **Spectrum Spatial for Big Data** documentation landing page.

Output Fields Variable

The output fields can use the variable `pb.geocoding.output.fields` set to a comma-separated list of fields. This list is case-sensitive and a mistyped field will return null. The above example can be created from the following statement.

```
set pb.geocoding.output.fields=X,Y,formattedStreetAddress,formattedLocationAddress;
```

PBKey and Other Custom Fields

Certain output fields will not be returned unless you edit the geocode preferences. To learn more, see **Geocode Preferences** on page 21.

Error Field

The error field is always returned and does not need to be requested. If the value is null then no errors occurred during the functions execution. See **Geocode-Level Errors** on page 22 for explanations of possible errors.

Spark Setup

To set up geocoding for Spark, perform the following steps from the same node used in **Installing the SDK and Configuring Reference Data** on page 7:

1. Copy the Spark geocoding jar to the node where you execute your job commands for Spark:

   ```
   /pb/geocoding/software/spark2/driver/spectrum-bigdata-geocoding-spark2-version-all.jar
   ```

2. Requested output fields can be set as a space-separated list of fields (see **Spark Output Fields** on page 18).

Spark Output Fields

The result of a Geocode or Reverse Geocode Spark Job is a CSV file with all columns from the input plus additional requested output fields. To change the output fields, you must use the `geocoding-output-fields` parameter.

> **Note:** For information on custom output fields per country, see the appendix in the ***Global Geocoding SDK Developer Guide***, which is also available on the **Spectrum Spatial for Big Data** documentation landing page.

Output Fields Parameter

The output fields can use the `geocoding-output-fields` parameter set to a space-separated list of fields. For example:

```
spark-submit
--class com.pb.bigdata.geocoding.spark.app.GeocodeDriver
--master yarn  --deploy-mode cluster
/pb/geocoding/software/spark2/driver/spectrum-bigdata-geocoding-spark2-version-all.jar
--input /user/pbuser/customers/addresses.csv
--output /user/pbuser/customers_geocoded
--geocoding-output-fields x y formattedStreetAddress formattedLocationAddress
--geocoding-resources-location hdfs:///pb/geocoding/software/resources/
--download-location /pb/downloads
--geocoding-preferences-filepath
hdfs:///pb/geocoding/software/resources/config/geocodePreferences.xml
--geocoding-input-fields streetName=0 areaName3=1 areaName1=2 postCode1=3
```

This list is case-sensitive and a mistyped field will return a column of null values.

PBKey and Other Custom Fields

Certain output fields will not be returned unless you edit the geocodePreferences.xml file. This file path is set using the `geocoding-preferences-filepath` command line parameter. To learn more, see **Geocode Preferences** on page 21.

Errors

All input records will go to output. If a row errors then it will be put in the `-output` location. In the case where no candidates are found, you will get null values for all geocoding output fields. Errors will also get null values for all geocoding output fields, and you can add the error field to your output to see any error information. See **Geocode-Level Errors** on page 22 for explanations of possible errors.

4 - Advanced Setup Options

In this section

Geocode Preferences 21
Geocode-Level Errors 22
Download Permissions 23

Geocode Preferences

You may need to change how the geocoder behaves or set some option that is not directly exposed. In order to do that you will need to edit the geocodePreferences.xml file located at `hdfs:///pb/geocoding/software/resources/config/geocodePreferences.xml`.

If you do not want to store this remotely (on HDFS or S3), then it must exist on all data nodes and master nodes.

PB_Key

Master Location Data datasets support a unique feature called PB_KEY. This is an identifier attached to every address record that facilitates data enrichment for faster and more informed decision making. To get PB_KEY, add the following XML element inside the geocodePreferences.xml file:

```
<returnAllAvailableCustomFields>true</returnAllAvailableCustomFields>
```

You will still need to request PB_KEY in the outputFields field.

Default Coordinate System for Reverse Geocoding

If you do not specify the coordinate system when performing a reverse geocode, then the default used is WGS84 (EPSG:4326). If you want a different default to be used, then set the following XML element inside the geocodePreferences.xml file:

```
<returnedPointCoordinateSystem>epsg:3857</returnedPointCoordinateSystem>
```

Geocode-Level Errors

These errors occur in the error field of an individual geocode.

Error	Solution
Geocoding candidate not found A location could not be found with the given address.	This can be solved in different ways • Verify the proper columns are being used in the geocode function. For example, the state column may improperly be in the city argument. If you have the majority of geocodes hitting this error then this is probably the cause. Consider revisiting the UDF page of the function you are trying to use. • The address may need to be more precise or cleaned up. Even after doing all that it will still be possible to have an address that cannot be geocoded.
Data not found The geocoding reference data is missing or misconfigured on at least one master or data node.	This can be solved in different ways • Verify that JsonDataConfig.json has paths properly pointing to the installed geocoding data dictionaries. Every node needs to be configured, and they all need to be configured the same way. The data dictionaries also have to exist on each data node and master node. • Verify you have data dictionaries that cover the country or region you are trying to geocode.
Invalid record This row is invalid or malformed.	If all addresses have this error then verify the column indexes match up with your data. Otherwise look into the following: • Verify the row has the correct amount of columns. • If the row has special characters, like the Delimiter, Quote, or Escape characters, verify the row is properly escaped.

Download Permissions

Setting the download permissions allows multiple services to download and update the downloaded data when required. You should have a common operating system group of which all the service users who need to download the data are part of. For example, if Hive and YARN jobs are required to download data and use the same download location, then both the Hive and YARN operating system users should be part of a common operating system group. The group of the download directory should be the common operating system group, one that has Read, Write, and Execute (775) permissions for the owner and group.

Your group should contain services and users that will run jobs in your cluster. You may skip services you will not use or do not have installed. Services include YARN, Hive, Zeppelin, and Hue.

You also should include all operating system users who will run jobs such as pbuser and *<myOtherUser>*.

1. Add the group.

   ```
   sudo groupadd pbdownloads
   ```

2. Add users to the group.

   ```
   sudo usermod -a -G pbdownloads hive
   sudo usermod -a -G pbdownloads yarn
   sudo usermod -a -G pbdownloads zeppelin
   sudo usermod -a -G pbdownloads hue
   sudo usermod -a -G pbdownloads pbuser
   sudo usermod -a -G pbdownloads <myOtherUser>
   ```

3. Using a window where no job is running, restart all the services whose operating system users were added to the new group.
4. Using a window where no job is running, restart the session of all the operating system users that were added to new group (for example, pbuser).
5. Update the group to the common operating system group and update permissions to 775 for the download directory specified in `pb.download.location` property.

   ```
   sudo chgrp pbdownloads /pb/downloads
   sudo chmod 775 /pb/downloads
   ```

Copyright