Table of Contents

GeoWave User Guide

§

GeoWave User Guide

Links

Site (http://ngageoint.github.io/geowave/)

PDF (https://s3.amazonaws.com/geowave/0.9.4/docs/userguide.pdf)

Javadoc (https://s3.amazonaws.com/geowave/latest/docs/apidocs/index.html)

GitHub (https://github.com/ngageoint/geowave)

Packages (https://s3.amazonaws.com/geowave/0.9.4/docs/packages.html)

What is GeoWave

GeoWave is a library for storage, index, and search of multi-dimensional data on top of a sorted key-value datastore. GeoWave includes specific tailored implementations that have advanced support for OGC spatial types (up to 3 dimensions), and both bounded and unbounded temporal values. Both single and ranged values are supported on all axes. GeoWave's geospatial support is built on top of the GeoTools extensibility model. This means that it can integrate natively with any GeoTools compatible project (such as GeoServer and UDig), and can ingest GeoTools compatible data sources. GeoWave provides out-of-the-box support for the Apache Accumulo (https://accumulo.apache.org) and Apache HBase (https://hbase.apache.org) distributed key/value stores.

GeoWave Features:

- Adds multi-dimensional indexing capability to Apache Accumulo and Apache HBase
- Adds support for geographic objects and geospatial operators to Apache Accumulo and Apache HBase
- Provides a GeoServer (http://geoserver.org/) plugin to allow geospatial data in Accumulo to be shared and visualized via OGC standard services
- Provides Map-Reduce input and output formats for distributed processing and analysis of geospatial data

GeoWave attempts to do for Accumulo and HBase as PostGIS does for PostgreSQL.

Origin

GeoWave was developed at the National Geospatial-Intelligence Agency (NGA) in collaboration with RadiantBlue Technologies (http://www.radiantblue.com/) and Booz Allen Hamilton (http://www.boozallen.com/). The government has unlimited rights (https://github.com/ngageoint/geowave/blob/master/NOTICE) and is releasing this software to increase the impact of government investments by providing developers with the opportunity to take things in new directions. The software use, modification, and distribution rights are stipulated within the Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0.html) license.

Intent

Pluggable Backend

GeoWave is intended to be a multidimensional indexing layer that can be added on top of any sorted key-value store. Accumulo was chosen as the initial target architecture and support for HBase has been added as well. Any datastore which allows prefix based range scans should be straightforward extensions.

Modular Design

The architecture itself is designed to be extremely extensible with most of the functionality units defined by interfaces, and with default implementations of these interfaces to cover most use cases. The intent is that the out of the box functionality should satisfy 90% of use cases but the modular architecture allows for easy feature extension as well as integration into other platforms.

Self-Describing Data

GeoWave stores the information needed to manipulate data (such as configuration and format), in the database itself. This allows software to programmatically interrogate all the data stored in a single or set of GeoWave instances without needing bits of configuration from clients, application servers, or other external stores.

Theory

This is a brief overview of the theory and concepts behind GeoWave. For an in-depth discussion, please see the GeoWave Developer Guide. (http://ngageoint.github.io/geowave/devguide.html#theory)

Spatial Index

GeoWave creates a spatial index to represent multi-dimensional data in a manner that can be reduced to a series of ranges on a 1 dimensional number line. Examples of these include:

- latitude, longitude
- latitude, longitude, time
- latitude, longitude, altitude, time
- feature vector1, feature vector 2 (...), feature vector n

This is due to the way big table based databases store the data – as a sorted set of key/value pairs.

The goal is to provide a property that ensures values close in n-dimensional space are still close in 1-dimensional space. There are a few reasons for this, but primarily it's so we can represent an n-dimensional range selector (bbox typically – but can be abstracted to a hyper-rectangle) as a smaller number of highly contiguous 1d ranges.

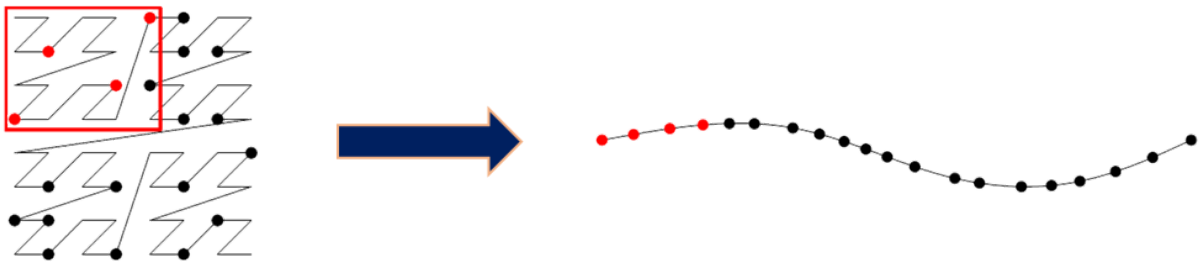


Figure: Z-Order curve based dimensional decomposition

Fortunately there is already a type of transform that describes this operation in mathematics – it's called a "Space Filling Curve" – or SFC for short. Different space filling curves have different properties, but they all take an n-dimensional space and describe a set of steps to trace all points in a single sequence.

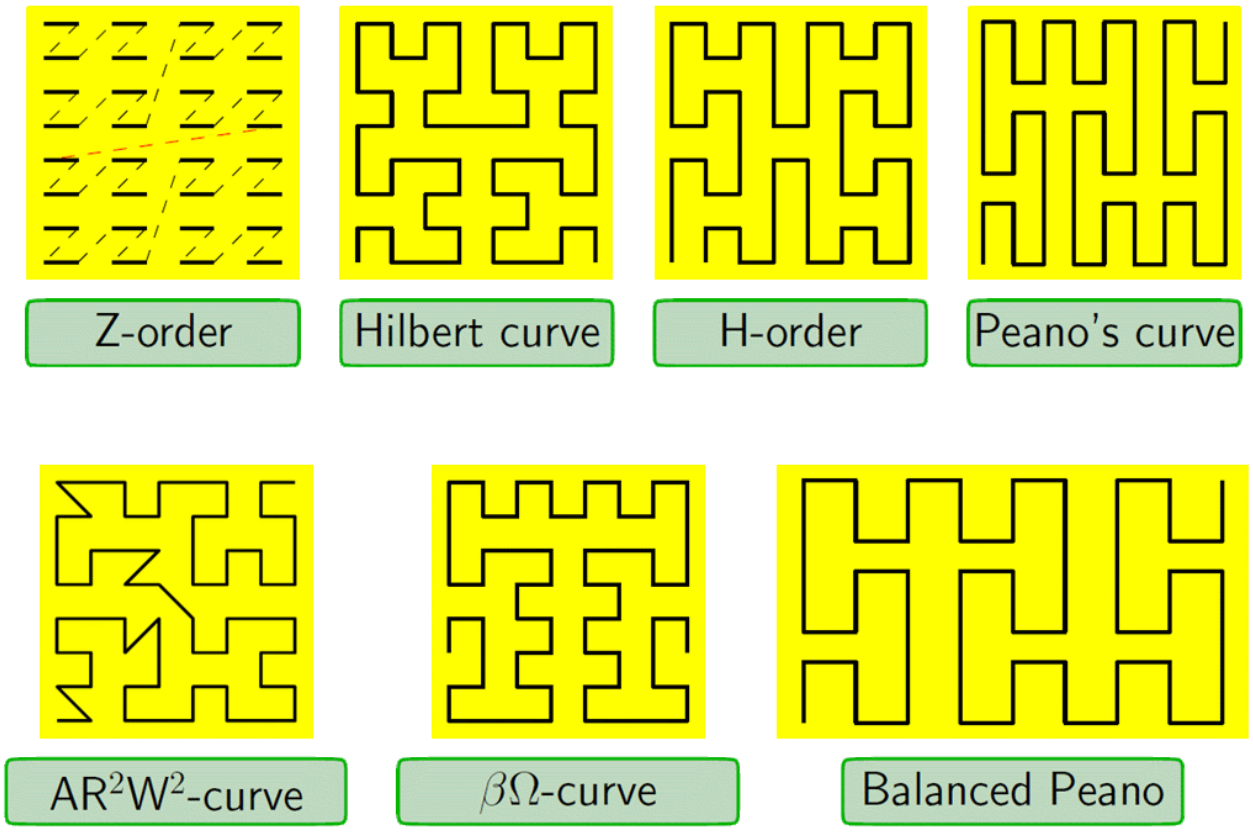


Figure: Haverkort, Walderveen Locality and Bounding-Box Quality of Two-Dimensional Space-Filling Curves 2008 arXiv:0806.4787v2

The trade-offs for the various curves are outside the scope of this user manual, but the paper cited for figure two is an excellent starting point to start learning about these curves.

GeoWave supports two space filling curves: Z-Order and Hilbert, with the Hilbert type space filling curve being the primary implementation.

Example screenshots

The screenshots below are of data loaded from various attributed data sets into a GeoWave instance, processed (in some cases) by a GeoWave analytic process, and rendered by Geoserver.

GeoLife

Microsoft research has made available a trajectory data set that contains the GPS coordinates of 182 users over a three year period (April 2007 to August 2012). There are 17,621 trajectories in this data set.

More information on this data set is available at Microsoft Research GeoLife page (http://research.microsoft.com/jump/131675)

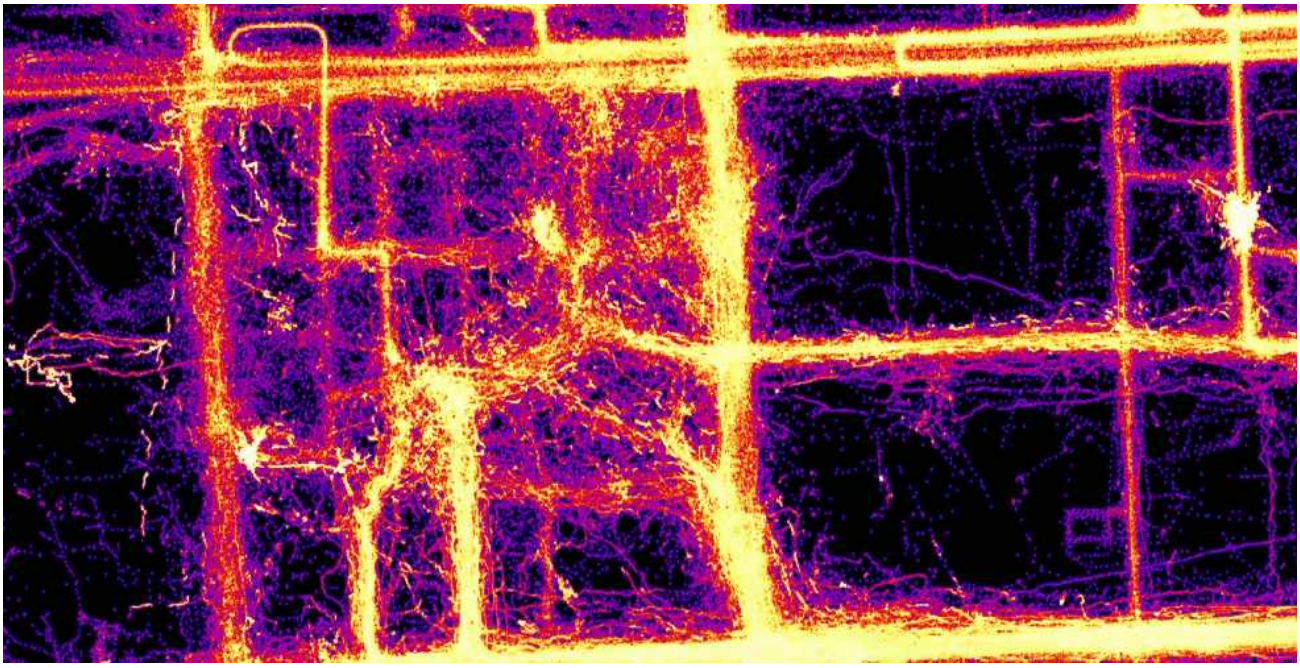
GeoLife at city scale

Below are renderings of GeoLife data. They display the raw points as well as the results of a GeoWave kernel density analytic. The data corresponds to Mapbox zoom level 13.

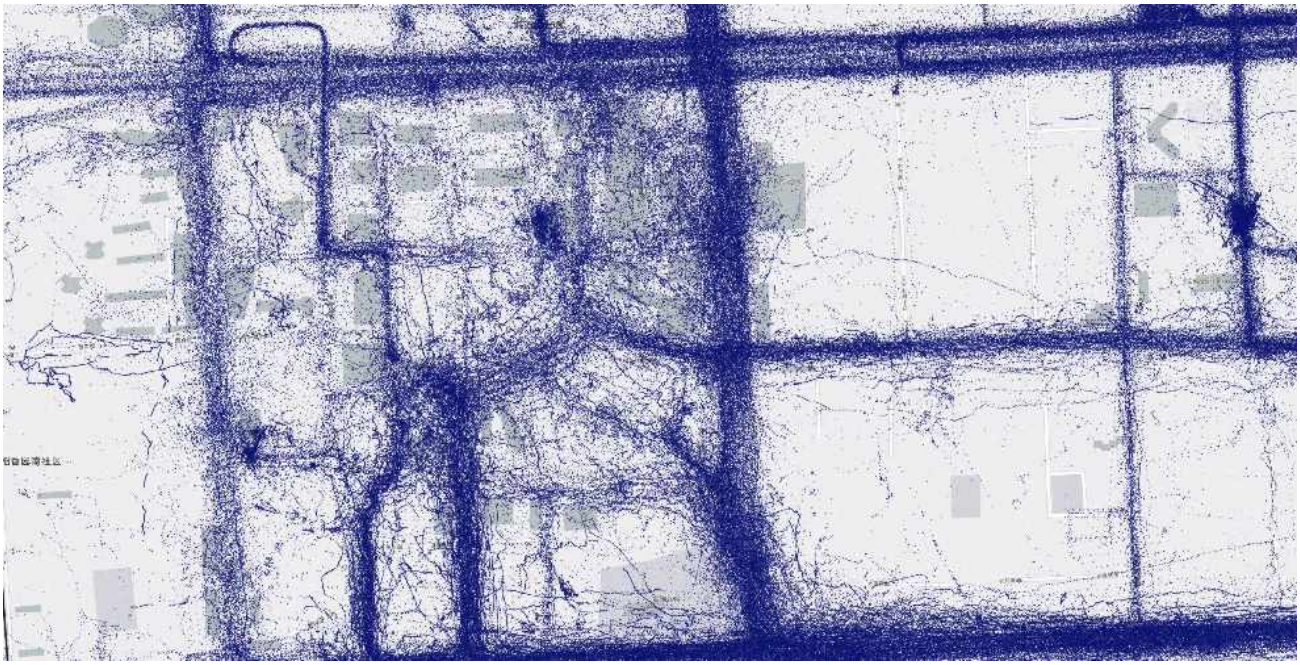
image::geolife-density-13-thumb.jpg[scaledwidth="100%",alt="Geolife density at city scale",link=images/geolife-density-13.jpg]
image::geolife-points-13-thumb.jpg[scaledwidth="100%",alt="Geolife points at city scale",link=images/geolife-points-13.jpg]

GeoLife at house scale

This data set corresponds to a Mapbox zoom level of 15



Graphic background ©MapBox and ©OpenStreetMap



Graphic background ©MapBox and ©OpenStreetMap

OpenStreetMap GPX Tracks

The OpenStreetMap Foundation has released a large set of user contributed GPS tracks. These are about 8 years of historical tracks. The data set consists of just under 3 billion (not trillion as some websites claim) points, or just under one million trajectories.

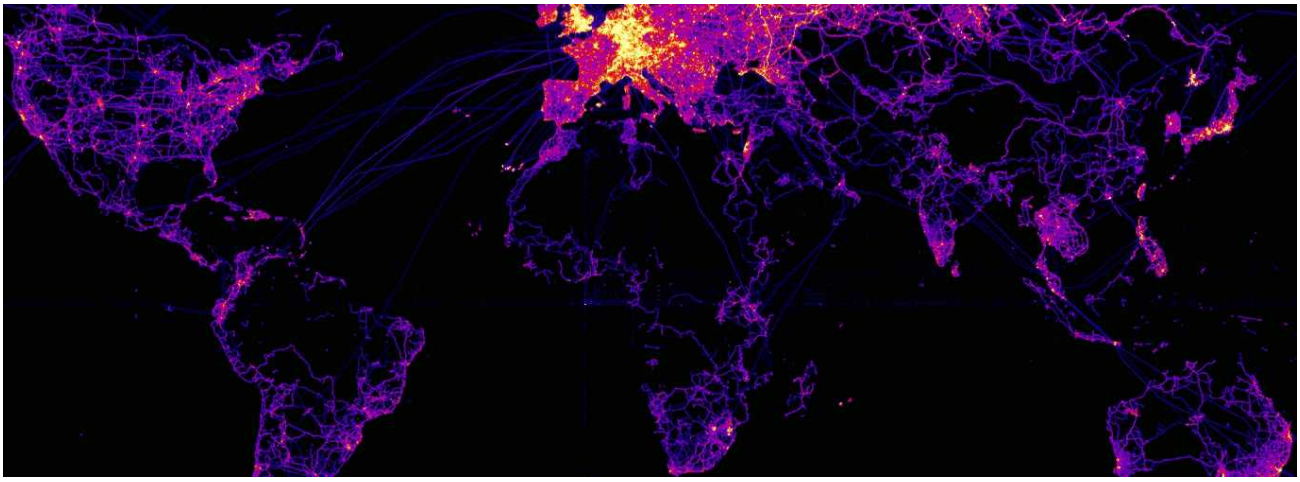
More information on this data set is available at GPX Planet page (http://wiki.openstreetmap.org/wiki/Planet.gpx)

OSM GPX at continent scale

The data below corresponds to a Mapbox zoom level of 6

OSM GPX at world scale

This data set corresponds to a Mapbox zoom level of 3



T-Drive

Microsoft research has made available a trajectory data set that contains the GPS coordinates of 10,357 taxis in Beijing, China and surrounding areas over a one week period. There are approximately 15 million points in this data set.

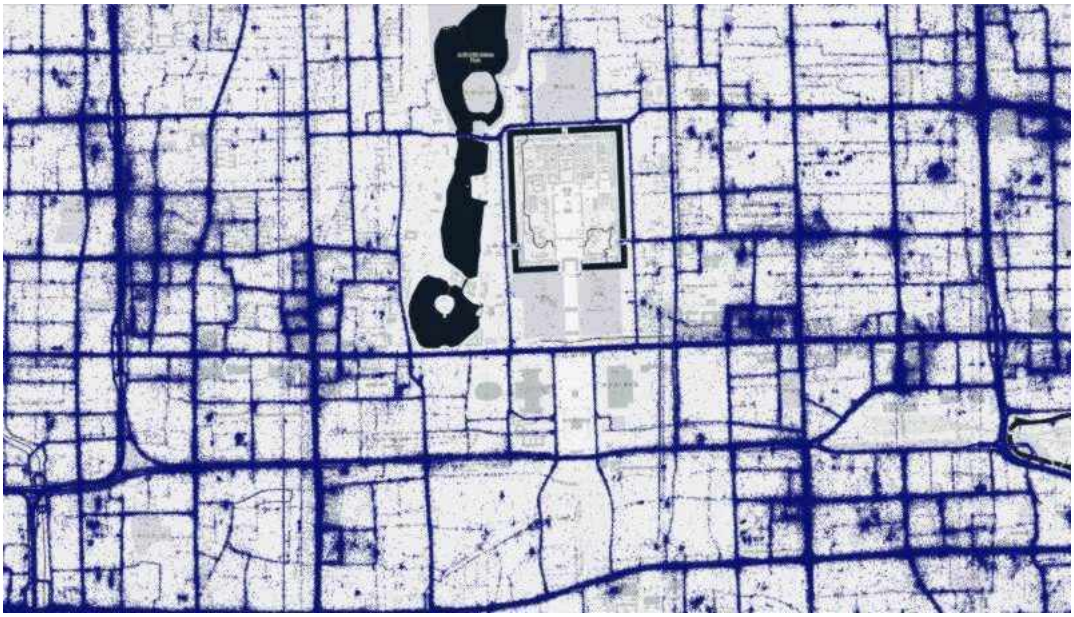
More information on this data set is available at: Microsoft Research T-drive page (http://research.microsoft.com/apps/pubs/?id=152883)

T-drive at city scale

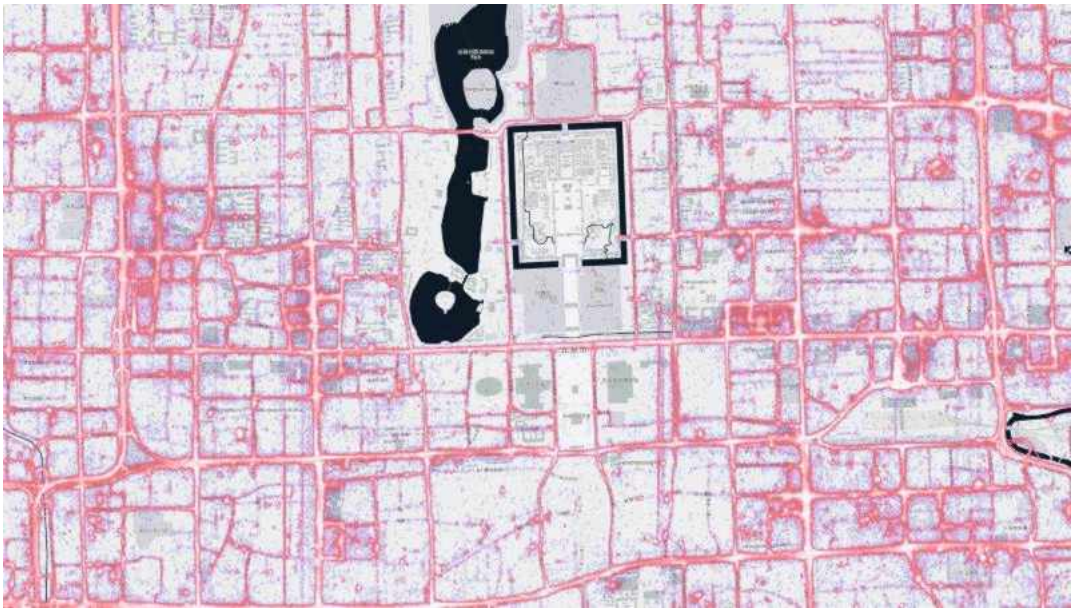
Below are renderings of the t-drive data. They display the raw points along with the results of a GeoWave kernel density analytic. The data corresponds to Mapbox zoom level 12.

T-drive at block scale

This data set corresponds to a Mapbox zoom level of 15



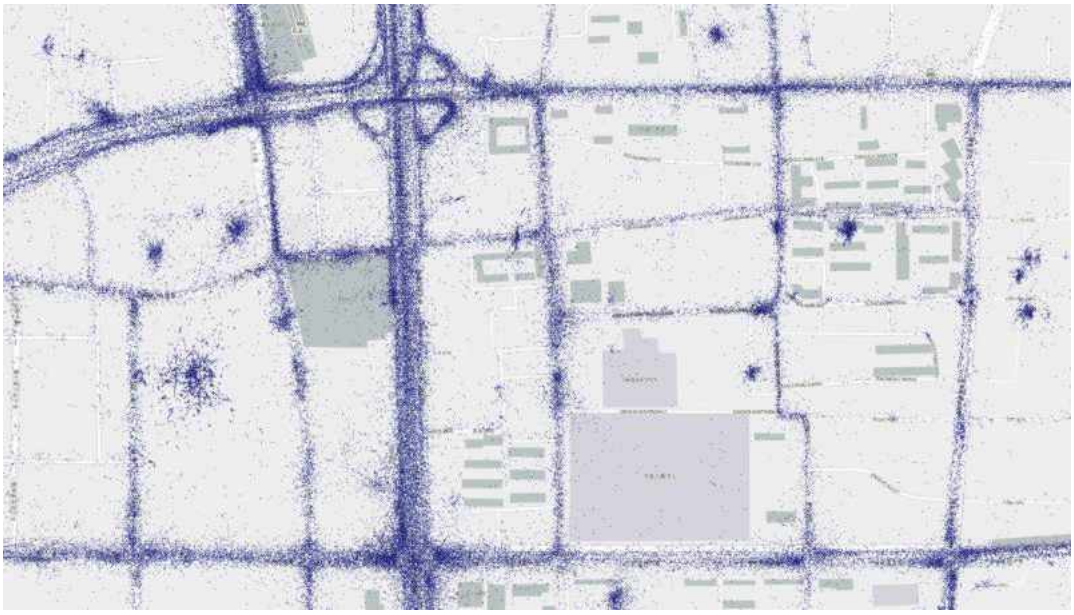
Graphic background©MapBox and ©OpenStreetMap



Graphic background©MapBox and ©OpenStreetMap

T-drive at house scale

This data set corresponds to a Mapbox zoom level of 17



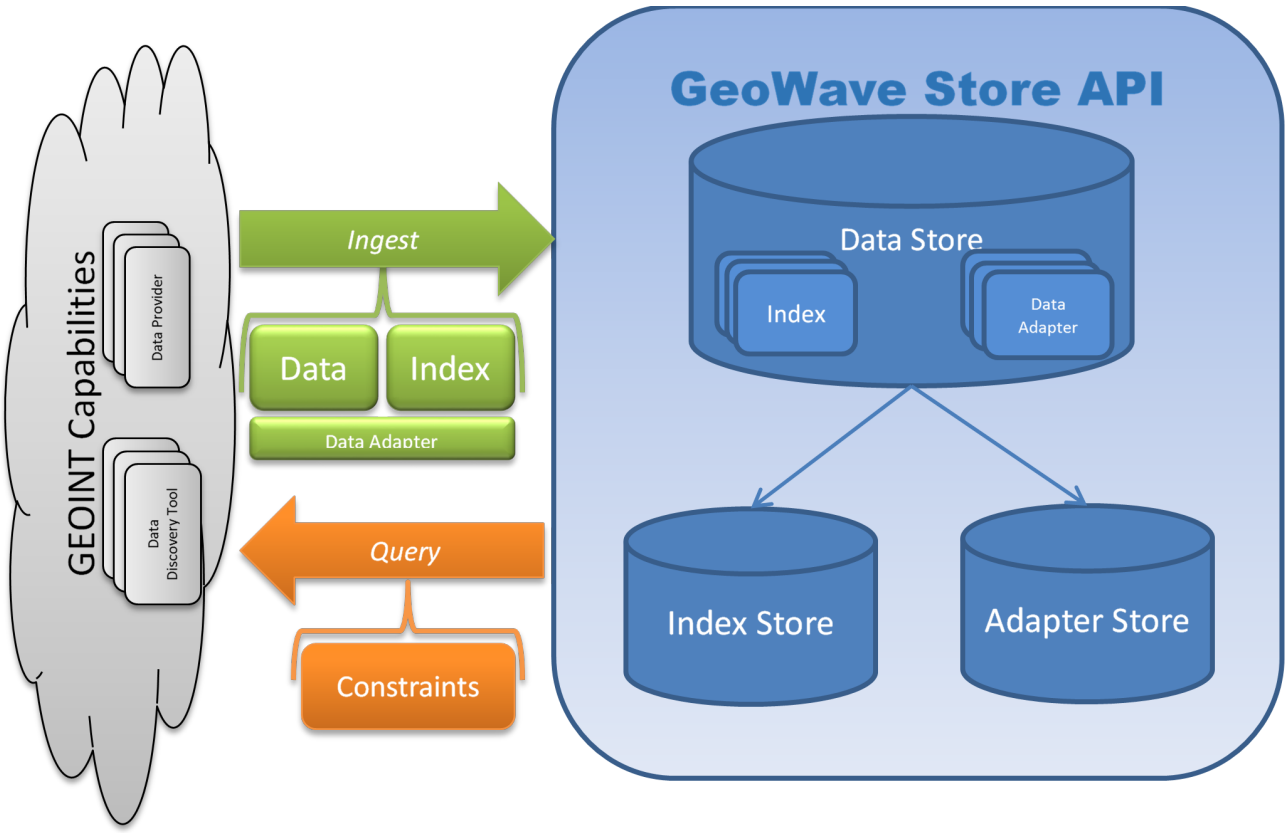
Graphic background©MapBox and ©OpenStreetMap

Graphic background©MapBox and ©OpenStreetMap

Architecture

This is a brief overview of the GeoWave architecture. For an in-depth discussion, please see the GeoWave Developer Guide. (http://ngageoint.github.io/geowave/devguide.html#architecture)

Overview



The core of the GeoWave architecture concept is getting data in, and pulling data out – or Ingest and Query. There are also two types of data persisted in the system: feature data, and metadata. Feature data is the actual set of attributes and geometries that are stored for later retrieval. Metadata describes how the data is persisted in the database. The intent is to store the information needed for data discovery and retrieval in the database. This means that an existing data store isn't tied to a bit of configuration on a particular external server or client, but instead is "self-describing."

Indexes

The core engine to quickly retrieve data from GeoWave is a SFC (space filling curve) based index. This index can be configured with several different parameters:

- number of levels
- number of dimensions
- cardinality of each dimension
- dimension type (bounded / unbounded)
- value range of each dimension

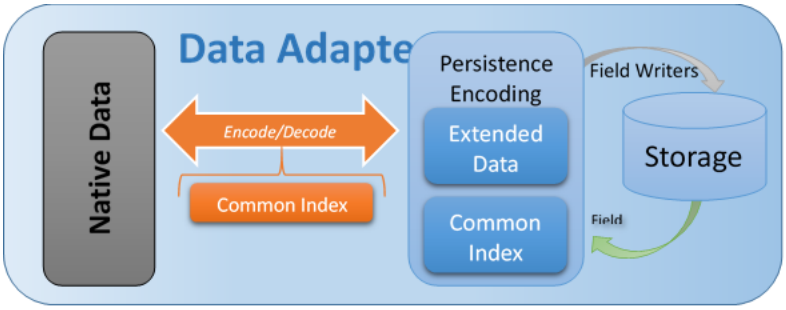
More on each of these properties will be described later; the intent of this list is to provide a notion of what type of configuration information is persisted.

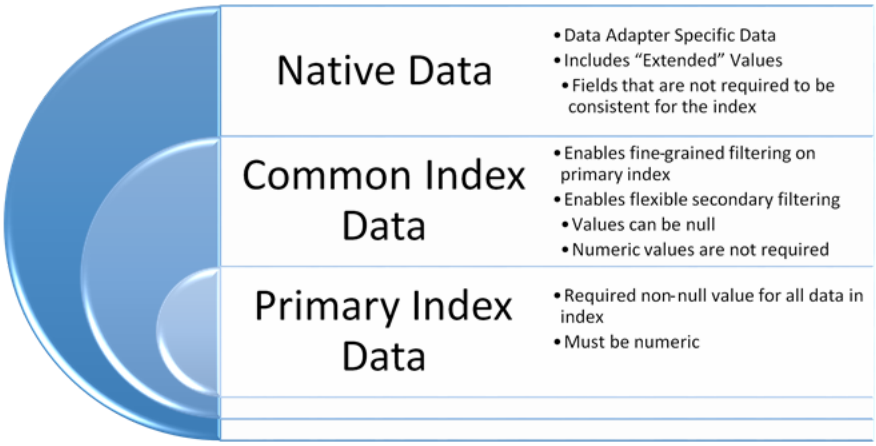
In order to insert data in a datastore the configuration of the index has to be known. The index is persisted in a special table and is referenced back via table name to a table with data in it. Therefore queries can retrieve data without requiring index configuration. There is a restriction that only one index configuration per table is supported - i.e. you can't store data on both a 2D and 3D index in the same table. (You could store 2D geometry types in a 3D index though).

Adapters

In order to store geometry, attributes, and other information, a format is required that describes how to serialize and deserialize the data. GeoWave provides an interface that handles the serialization and deserialization of feature data. A default implementation supporting the GeoTools simple feature type is included by default. More on this specific implementation, as well as the interface, will be detailed later in this document. A pointer to the java class (expected to be on the classpath) is stored in the Adapter persistence table. This is loaded dynamically when the data is queried and the results are translated to the native data type

Feature Serialization





GeoWave allows developers to create their own data adapters. Adapters not only determine how the data is actually stored (serialization/deserialization), but also contain a hierarchy of attribute types. The reason for this hierarchy has to do with extensibility vs. optimization. A data adapter could theoretically take a dependency on ffmpeg, store the feature as metadata in a video stream, and persist that value to the database. All questions of sanity of this solution aside, there are some additional specific issues with the way fine grain filtering is done - specifically due to the iterators. Iterators are part of the Accumulo extensibility module and allow for arbitrary code to be plugged in directly at the tablet server level into the core Accumulo kernel. With more code in the iterators there is both a greater chance of crashing (and taking down a tablet server - and possibly an Accumulo instance), greater use of memory (memory used by the iterator / class loader isn't available for caching, etc., and a greater O&M debt - the iterators have to be distributed to each client out of band - and require impersonating the Accumulo user (possibly root).

Based on this, our goal was to minimize the code, and standardize on as few iterators as possible. This conflicted with the desire to allow maximum flexibility with arbitrary DataAdapters. A middle ground was found, and this hierarchy was created. Some standardization was put in place around how certain data types would be stored and serialized, but a "native data" portion was still left in place for arbitrary data - with the caveat that native data cannot be used in distributed (iterator based) filtering - only in client side filtering.

Primary Index Data

These are sets of data which are also used to construct the primary index (space filling curve). They will typically be geometry coordinates and optionally time - but could be any set of numeric values (think decomposed feature vectors, etc.). They cannot be null.

Common Index Data

These are a collection of attributes. There can be any number of attributes, but they must conform to the DimensionField interface - the attribute type must have a FieldReader and a FieldWriter that is within the classpath of the tablet servers. GeoWave provides a basic implementation for these attribute types:

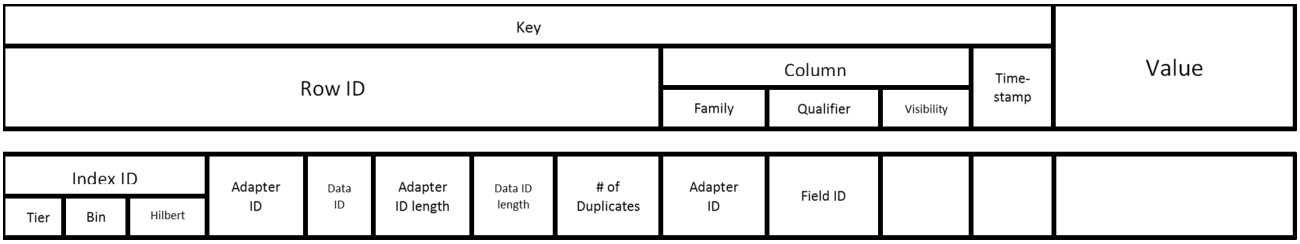
- Boolean
- Byte
- Short
- Float
- Double
- BigDecimal
- Integer
- Long
- BigInteger
- String
- Geometry
- Date
- Calendar

The values that are not part of the primary index can be used for distributed secondary filtering, and can be null. The values that are associated with the primary index will be used for fine-grained filtering within an iterator.

Native Data

These can be literally anything. From the point of view of the data adapter they are just a binary (or Base64) encoded chunk of data. No distributed filtering can be performed on this data except for Accumulo's visibility filter - but the client side filtering extensibility point can still be used if necessary. The Data Adapter has to provide methods to serialize and deserialize these items in the form of Field Readers and Writers, but it is not necessary to have these methods on the classpath of any nodes.

Key Structure



The above diagram describes the default structure of entries in the data store. The index ID comes directly from the tiered space filling curve implementation. We do not impose a requirement that data IDs are globally unique but they should be unique for the adapter. Therefore, the pairing of Adapter ID and Data ID define a unique identifier for a data element. The lengths are stored within the row ID as 4 byte integers. This enables fully reading the row ID because these IDs can be of variable length. The number of duplicates is stored within the row ID as well to inform the de-duplication filter whether this element needs to be temporarily stored in order to ensure no duplicates are sent to the caller. The adapter ID is within the Row ID to enforce unique row IDs as a whole row iterator is used to aggregate fields for the distributable filters. The adapter ID is also used as the column family as the mechanism for adapter-specific queries to fetch only the appropriate column families.

Statistics

Adapters provide a set of statistics stored within a statistic store. The set of available statistics is specific to each adapter and the set of attributes for those data items managed by the adapter. Statistics include:

- Ranges over an attribute, including time.
- Enveloping bounding box over all geometries.
- Cardinality of the number of stored items.
- Histograms over the range of values for an attribute.
- Cardinality of discrete values of an attribute.

Statistics are updated during data ingest and deletion. Range and bounding box statistics reflect the largest range over time. Those statistics are not updated during deletion. Statistics based on cardinality are updated upon deletion.

Tools Framework

A plugin framework (using Service Provider Interface (SPI) based injection) is provided with several input formats and utilities supported out of the box.

First we'll show how to build and use the built in formats, and after that describe how to create a new plugin.

Building

First build the main project after specifying the dependency versions you'd like to build against.

```bash
export BUILD_ARGS="-Daccumulo.version=1.7.2 -Daccumulo.api=1.7 -Dhbase.version=1.3.0 -Dhadoop.version=2.7.3 -
Dgeotools.version=16.0 -Dgeoserver.version=2.10.0" (1)
git clone https://github.com/ngageoint/geowave.git (2)
cd geowave
mvn install $BUILD_ARGS (3)
```

(1) Examples of current build args can be seen in the top level .travis.yml file in the env/matrix section

(2) If you don't need the complete history and want to speed up the clone you can limit the depth of your checkout with --depth NUM_COMMITS

(3) You can speed up the build by skipping tests by adding -Dfindbugs.skip=true -Dformatter.skip=true -DskipITs=true -DskipTests=true

Now we can build the cli tools framework

```bash
mvn package -P geowave-tools-singlejar $BUILD_ARGS
```

The geowave tools jar is now packaged in deploy/target. When packaged for installation there will be a wrapper script named geowave that will be installed in $PATH. In a development environment where this script has not been installed you could create a directory containing the tools jar and any needed plugin jars and use with something like the following command `java -cp "$DIR/* <operation> <options>`

At this point you can now run GeoWave Command Line Instructions. For a full list of these commands please see the GeoWave CLI Appendix (http://ngageoint.github.io/geowave/commands.html).

Ingest

Overview

In addition to the raw data to ingest, the ingest process requires an adapter to translate the native data into a format that can be persisted into the data store. Also, the ingest process requires an Index which is a definition of all the configured parameters that define how data is translated to row IDs (how it is indexed) and what common fields need to be maintained within the table to be used by fine-grained and secondary filters.

The full list of GeoWave ingest commands can be found in the GeoWave CLI Appendix (http://ngageoint.github.io/geowave/commands.html#ingest-commands).

Ingest Example

GeoWave can ingest any data type that has been listed as an ingest plugin. Let's start out with the GeoTools datastore; this wraps a bunch of GeoTools supported formats. This includes all file-based datastores supported within GeoTools. We will use the shapefile capability for our example here.

Something recognizable

The naturalearthdata side has a few shapefile we can use use. On the page 50m Cultural Vectors (http://www.naturalearthdata.com/downloads/50m-cultural-vectors/)

Let's download the Admin 0 - Countries shapefile: ne_50m_admin_0_countries.zip (http://naciscdn.org/naturalearth/50m/cultural/ne_50m_admin_0_countries.zip)

BASH

```
$ mkdir ingest
$ mv ne_50m_admin_0_countries.zip ingest/
$ cd ingest
$ unzip ne_50m_admin_0_countries.zip
$ rm ne_50m_admin_0_countries.zip
$ cd ..
$ geowave config addstore -t accumulo mystore \ (1)
      --gwNamespace geowave.50m_admin_0_countries \ (2)
      --zookeeper ${ZOOKEEPER_HOSTNAME}:2181 \ (3)
      --instance ${ACCUMULO_INSTANCE_NAME} \
      --user ${USERNAME} \
      --password ${PASSWORD}
$ geowave config addindex -t spatial myindex \ (4)
      --partitionStrategy round_robin \
          --numPartitions ${NUM_PARTITIONS} \ (5)
$ geowave ingest localToGW ./ingest mystore myindex \ (6)
          -f geotools-vector \ (7)
```

1. This will create a re-usable named configuration `mystore` that can be referenced as a store by other commandline operations such as ingest. The type (-t) can also be set to hbase if you prefer to use HBase as your datastore
2. We preface the table name with the Accumulo namespace we configured earlier in Accumulo followed by a dot (NAMESPACE.TABLE_NAME). The Accumulo Configuration section can explain this further.
3. If you are using an HBase datastore, zookeeper is the only required option here
4. This will create a re-usable named configuration `myindex` that can be referenced as an index by other commandline operations such as ingest
5. The index is spatial and pre-split based on the number of partitions you may desire - this is an optional parameter but an example of customization you may choose on index configuration, in this case data is randomized into different splits which can help avoid hotspotting to a single node/core
6. Notice the ingest command uses the named configurations `mystore` and `myindex`
7. Explicitly set the ingest formats by name (or multiple comma-delimited formats), if not set all available ingest formats will be used

After running the ingest command you should see the various index tables in you datastore.

Ingest Plugins

The geowave command line utility comes with several plugins out of the box. You can list the available plugins that are registered with your commandline tool.

```
geowave ingest listplugins
```

You can add more by simply copying a desired plugin into the /usr/local/geowave/tools/plugins directory.

Available index types currently registered as plugins:

- spatial_temporal
  - This dimensionality type matches all indices that only require Geometry and Time.
- spatial
  - This dimensionality type matches all indices that only require Geometry.

Available ingest formats currently registered as plugins:

- geotools-vector
  - all file-based vector datastores supported within geotools
- geolife
  - files from Microsoft Research GeoLife trajectory data set
- gdelt
  - files from Google Ideas GDELT data set
- stanag4676
  - xml files representing track data that adheres to the schema defined by STANAG-4676
- geotools-raster
  - all file-based raster formats supported within geotools
- gpx
  - xml files adhering to the schema of gps exchange format
- tdrive
  - files from Microsoft Research T-Drive trajectory data set
- avro
  - This can read an Avro file encoded with the SimpleFeatureCollection schema. This schema is also used by the export tool, so this format handles re-ingesting exported datasets.

Available datastores currently registered:

- accumulo
  - A GeoWave store backed by tables in Apache Accumulo
- hbase
  - A GeoWave store backed by tables in Apache HBase

Ingest Statistics and Time Dimension Configuration

The available plugins for vector support adjustments to their configuration via the command line. The system property 'SIMPLE_FEATURE_CONFIG_FILE' may be assigned to the name of a locally accessible JSON file defining the configuration.

Example

```
$ GEOWAVE_TOOL_JAVA_OPT="-DSIMPLE_FEATURE_CONFIG_FILE=myconfigfile.json"
$ geowave ingest localtogw ./ingest mystore myindex
```

Configuration consists of several parts:

1. Selecting temporal attributes for a temporal index.
2. Assigning to each attribute the type of statistics to be captured within the Statistics Store
3. Determining which attributes should be indexed in a secondary index.
4. Determining which attribute contains visibility information for other attributes
5. Setting the names of the indices to update in WFS-T transactions via the GeoServer plug-in.

The JSON file is made up of configurations. Each configuration is defined by a class name and a set of attributes. Configurations are grouped by the Simple Feature Type name.

Temporal Configuration

There are three attributes for the temporal configuration:

1. timeName
2. startRangeName
3. endRangeName

These attributes are associated with the name of a simple feature type attribute that references a time value. To index by a single time attribute, set `timeName` to the name of the single attribute. To index by a range, set both `startRangeName` and `endRangeName` to the names of the simple feature type attributes that define start and end time values.

Statistics Configuration

Each simple feature type attribute may have several assigned statistics. Bounding box and range statistics are automatically captured for geometry and temporal attributes.

Attribute Type	Statistic Name	Statistic Configuration Attributes (with default values)	Statistic Class
Numeric	Fixed Bin Histogram	minValue=-∞,maxValue=∞,bins=32	mil.nga.giat.geowave.adapter.vector.stats.FeatureFixedBinNumericStatistics$FeatureFixedBinConfig
	Dynamic Histogram		mil.nga.giat.geowave.adapter.vector.stats.FeatureNumericHistogramStatistics$FeatureNumericHistogra
	Numeric Range		mil.nga.giat.geowave.adapter.vector.stats.FeatureNumericRangeStatistics$FeatureNumericRangeConfig
String	Count Min Sketch	errorFactor=0.001,probabilityOfCorrectness=0.98	mil.nga.giat.geowave.adapter.vector.stats.FeatureCountMinSketchStatistics$FeatureCountMinSketchCor
	Hyper Log Log	precision=16	mil.nga.giat.geowave.adapter.vector.stats.FeatureHyperLogLogStatistics$FeatureHyperLogLogConfig

Visibility Configuration

Visibility configuration has two attributes: the visibility manager class and the visibility attribute name.

A Visibility manager extends mil.nga.giat.geowave.core.store.data.visibility.VisibilityManagement. An instance of this class interprets the contents of a visibility attribute, within a simple feature, to determine the visibility constraints of the other attributes in that simple feature. The default visibility management class is mil.nga.giat.geowave.adapter.vector.plugin.visibility.JsonDefinitionColumnVisibilityManagement.

Secondary Index Configuration

Secondary Index Configurations is made up of one of three classes: .
mil.nga.giat.geowave.adapter.vector.index.NumericSecondaryIndexConfiguration .
mil.nga.giat.geowave.adapter.vector.index.TemporalSecondaryIndexConfiguration .

mil.nga.giat.geowave.adapter.vector.index.TextSecondaryIndexConfiguration

Each of this configurations maintains a set of simple feature attribute names to index in a secondary index.

Primary Index Identifiers

The class mil.nga.giat.geowave.adapter.vector.index.SimpleFeaturePrimaryIndexConfiguration is used to maintain the configuration of primary indices used for adding or updating simple features via the GeoServer plug-in (FeatureWriter).

Example

```
{
  "configurations": {
     "myFeatureTypeName" : [
       {"@class":"mil.nga.giat.geowave.adapter.vector.utils.TimeDescriptors$TimeDescriptorConfiguration",
        "startRangeName":null,
        "endRangeName":null,
        "timeName":"captureTime"
       },
       { "@class": "mil.nga.giat.geowave.adapter.vector.index.NumericSecondaryIndexConfiguration",
         "attributes" : ["pop"]
       },
       { "@class": "mil.nga.giat.geowave.adapter.vector.plugin.visibility.VisibilityConfiguration",
         "attributeName" : "vis"
       },
       { "@class": "mil.nga.giat.geowave.adapter.vector.index.SimpleFeaturePrimaryIndexConfiguration",
         "indexNames": ["SPATIAL_IDX"]
       }

{"@class":"mil.nga.giat.geowave.adapter.vector.stats.StatsConfigurationCollection$SimpleFeatureStatsConfigurationCollecti
on",
        "attConfig" : {
          "population" : {
            "configurationsForAttribute" :  [
              {"@class" :
"mil.nga.giat.geowave.adapter.vector.stats.FeatureFixedBinNumericStatistics$FeatureFixedBinConfig","bins" : 24}
            ]
            },
          "country" : {
            "configurationsForAttribute" :  [
             {"@class" :
"mil.nga.giat.geowave.adapter.vector.stats.FeatureCountMinSketchStatistics$FeatureCountMinSketchConfig",
              "probabilityOfCorrectness" : 0.98,
              "errorFactor" :0.001
             },
             {"@class" :
"mil.nga.giat.geowave.adapter.vector.stats.FeatureHyperLogLogStatistics$FeatureHyperLogLogConfig"}
            ]
          }
        }
      }
    ]
  }
}
```

New Formats

There are multiple ways to get data into GeoWave. In other sections we will discuss higher order frameworks, mapreduce interfaces, etc. The intent here is "just the basics" - the least framework intensive way that one can load geospatial data.

Information here will reference the SimpleIngest and SimpleIngestProducerConsumer examples in the geowave-examples project.

The following example is using Accumulo for our datastore. Using HBase for the datastore would work the same way, except you would use the BasicHBaseOperations object instead of the BasicAccumuloOperations object.

Minimum information needed

Geowave requires a few pieces of fundamental information in order to persist data - these are:

- BasicAccumuloOperations object
- This class contains the information required to connect to an accumulo instance - and which table to use in accumulo.

- Zookeepers - in the format zookeeper1:port,zookeeper2:port,etc...
- Accumulo Instance ID - this is the "instance" that the Accumulo cluster you are connecting to was initialized with. It's a global setting per cluster.
- Accumulo Username - this is the name of the user you would like to connect as. This is a user account managed by accumulo, not a system, etc. user.
- Accumulo Password - this is the password associated with the user specified above. Again, this is an accumulo controlled secret.
- Geowave Namespace - this is *not* an Accumulo namespace; rather think of it as a prefix geowave will use on any tables it creates. The only current constraint is only one index type is allowed per namespace.

- SimpleFeatureType instance
- Simple Feature Types (http://www.opengeospatial.org/standards/sfs) are an OGC specification for defining geospatial features. Leveraging this standard is one of the easiest ways to get GIS data into GeoWave
- SimpleFeatureType instance - org.opengis.feature.simple.SimpleFeatureType - this defines the names, types, and other metadata (nullable, etc) of a feature. Think of it as a Map of Name:Values where the values are typed.
- DataAdapter instance
- A geowave data adapter is an implementation of the DataAdapter interface that handles the persistence serialization of whatever the object you are storing.
- We are storing SimpleFeatures, so can leverage the provided FeatureDataAdapter
- Index instance
- The final piece needed - the index defines which attributes are indexed, and how that index is constructed.
- There are lots of options for index configuration, but for convenience we have provided two defaults
- DataStore
- This is the piece that puts everything above together.
- Initialization required a BasicAccumuloOperations instance, the rest are provided as parameters for calls which need them.

Ingest some data

Here we will programmatically generate a grid of points at each location where a whole number latitude and longitude intersect.

Basic Accumulo Operations

JAVA

```
/***
 * The class tells geowave about the accumulo instance it should connect to, as well as what tables it should
create/store it's data in
 * @param zookeepers Zookeepers associated with the accumulo instance, comma separate
 * @param accumuloInstance  Accumulo instance name
 * @param accumuloUser   User geowave should connect to accumulo as
 * @param accumuloPass   Password for user to connect to accumulo
 * @param geowaveNamespace    Different than an accumulo namespace (unfortunate naming usage) - this is basically a
prefix on the table names geowave uses.
 * @return  Object encapsulating the accumulo connection information
 * @throws AccumuloException
 * @throws AccumuloSecurityException
 */
protected BasicAccumuloOperations getAccumuloInstance(String zookeepers, String accumuloInstance, String accumuloUser,
String accumuloPass, String geowaveNamespace)
  throws AccumuloException, AccumuloSecurityException {
    return new BasicAccumuloOperations(zookeepers, accumuloInstance, accumuloUser, accumuloPass, geowaveNamespace);
}
```

Simple Feature Type

A geometry field is required. Everything else is really optional. It's often convenient to add a text latitude and longitude field for ease of display values (getFeatureInfo, etc.).

```java
/***
 * A simple feature is just a mechanism for defining attributes (a feature is just a collection of attributes + some metadata)
 * We need to describe what our data looks like so the serializer (FeatureDataAdapter for this case) can know how to store it.
 * Features/Attributes are also a general convention of GIS systems in general.
 * @return Simple Feature definition for our demo point feature
 */
protected SimpleFeatureType createPointFeatureType(){

    final SimpleFeatureTypeBuilder builder = new SimpleFeatureTypeBuilder();
    final AttributeTypeBuilder ab = new AttributeTypeBuilder();


    //Names should be unique (at least for a given GeoWave namespace) - think about names in the same sense as a full classname
    //The value you set here will also persist through discovery - so when people are looking at a dataset they will see the
    //type names associated with the data.
    builder.setName("Point");

    //The data is persisted in a sparse format, so if data is nullable it will not take up any space if no values are persisted.
    //Data which is included in the primary index (in this example lattitude/longtiude) cannot be null
    //Calling out latitude an longitude separately is not strictly needed, as the geometry contains that information. But it's
    //convienent in many use cases to get a text representation without having to handle geometries.
    builder.add(ab.binding(Geometry.class).nillable(false).buildDescriptor("geometry"));
    builder.add(ab.binding(Date.class).nillable(true).buildDescriptor("TimeStamp"));
    builder.add(ab.binding(Double.class).nillable(false).buildDescriptor("Latitude"));
    builder.add(ab.binding(Double.class).nillable(false).buildDescriptor("Longitude"));
    builder.add(ab.binding(String.class).nillable(true).buildDescriptor("TrajectoryID"));
    builder.add(ab.binding(String.class).nillable(true).buildDescriptor("Comment"));

    return builder.buildFeatureType();
}
```

Spatial index

```java
/***
 * We need an index model that tells us how to index the data - the index determines
 * -What fields are indexed
 * -The precision of the index
 * -The range of the index (min/max values)
 * -The range type (bounded/unbounded)
 * -The number of "levels"  (different precisions, needed when the values indexed has ranges on any dimension)
 * @return GeoWave index for a default SPATIAL index
 */
protected Index createSpatialIndex(){

    //Reasonable values for spatial and spatio-temporal are provided through static factory methods.
    //They are intended to be a reasonable starting place - though creating a custom index may provide better
    //performance is the distribution/characterization of the data is well known.
    return IndexType.SPATIAL.createDefaultIndex();
}
```

Data Adapter

```java
/***
 * The dataadapter interface describes how to serialize a data type.
 * Here we are using an implementation that understands how to serialize
 * OGC SimpleFeature types.
 * @param sft  simple feature type you want to generate an adapter from
 * @return data adapter that handles serialization of the sft simple feature type
 */
protected FeatureDataAdapter createDataAdapter(SimpleFeatureType sft){
    return new FeatureDataAdapter(sft);
}
```

Generating and loading points

```java
protected void generateGrid(
            final BasicAccumuloOperations bao ) {

        // create our datastore object
        final DataStore geowaveDataStore = getGeowaveDataStore(bao);

        // In order to store data we need to determine the type of data store
        final SimpleFeatureType point = createPointFeatureType();

        // This a factory class that builds simple feature objects based on the
        // type passed
        final SimpleFeatureBuilder pointBuilder = new SimpleFeatureBuilder(
                point);

        // This is an adapter, that is needed to describe how to persist the
        // data type passed
        final FeatureDataAdapter adapter = createDataAdapter(point);

        // This describes how to index the data
        final Index index = createSpatialIndex();

        // features require a featureID - this should be unqiue as it's a
        // foreign key on the feature
        // (i.e. sending in a new feature with the same feature id will
        // overwrite the existing feature)
        int featureId = 0;

        // get a handle on a GeoWave index writer which wraps the Accumulo
        // BatchWriter, make sure to close it (here we use a try with resources
        // block to close it automatically)
        try (IndexWriter indexWriter = geowaveDataStore.createIndexWriter(index)) {
            // build a grid of points across the globe at each whole
            // lattitude/longitude intersection
            for (int longitude = -180; longitude <= 180; longitude++) {
                for (int latitude = -90; latitude <= 90; latitude++) {
                    pointBuilder.set(
                            "geometry",
                            GeometryUtils.GEOMETRY_FACTORY.createPoint(new Coordinate(
                                    longitude,
                                    latitude)));
                    pointBuilder.set(
                            "TimeStamp",
                            new Date());
                    pointBuilder.set(
                            "Latitude",
                            latitude);
                    pointBuilder.set(
                            "Longitude",
                            longitude);
                    // Note since trajectoryID and comment are marked as
                    // nillable we
                    // don't need to set them (they default ot null).

                    final SimpleFeature sft = pointBuilder.buildFeature(String.valueOf(featureId));
                    featureId++;
                    indexWriter.write(
                            adapter,
                            sft);
                }
            }
        }
        catch (final IOException e) {
            log.warn(
                    "Unable to close index writer",
                    e);
        }
    }
```

Other methods

There are other patterns that can be used. See the various classes in the geowave-examples project. The method displayed above is the suggested pattern - it's demonstrated in SimpleIngestIndexWriter.java

The other methods displayed work, but are either more complicated than necessary (SimpleIngestProducerConsumer.java) or not very efficient (SimpleIngest.java).

Analytics

Overview

Analytics embody algorithms tailored to geospatial data. Most analytics leverage Hadoop MapReduce for bulk computation. Results of analytic jobs consist of vector or raster data stored in GeoWave. The analytics infrastructure provides tools to build algorithms in Spark. For example, a Kryo serializer/deserializer enables exchange of SimpleFeatures and the GeoWaveInputFormat supplies data to the Hadoop RDD

GeoWaveInputFormat does not remove duplicate features that reference polygons spanning multiple index regions.

The following algorithms are provided.

Name	Description
KMeans++	A K-Means implementation to find K centroids over the population of data. A set of preliminary sampling iterations find an optimal value of K and the an initial set of K centroids. The algorithm produces K centroids and their associated polygons. Each polygon represents the concave hull containing all features associated with a centroid. The algorithm supports drilling down multiple levels. At each level, the set centroids are determined from the set of features associated the same centroid from the previous level.
KMeans Jump	Uses KMeans++ over a range of k, choosing an optimal k using an information theoretic based measurement.
KMeans Parallel	Performs a KMeans Parallel Cluster
DBScan	The Density Based Scanner algorithm produces a set of convex polygons for each region meeting density criteria. Density of region is measured by a minimum cardinality of enclosed features within a specified distance from each other.
Nearest Neighbors	A infrastructure component that produces all the neighbors of a feature within a specific distance.

Building

Build the geowave tools project, as explained in the "ToolsFramework → Building" section.

Running

```
yarn jar geowave-tools.jar analytic <algorithm> <options> <store>
```
BASH

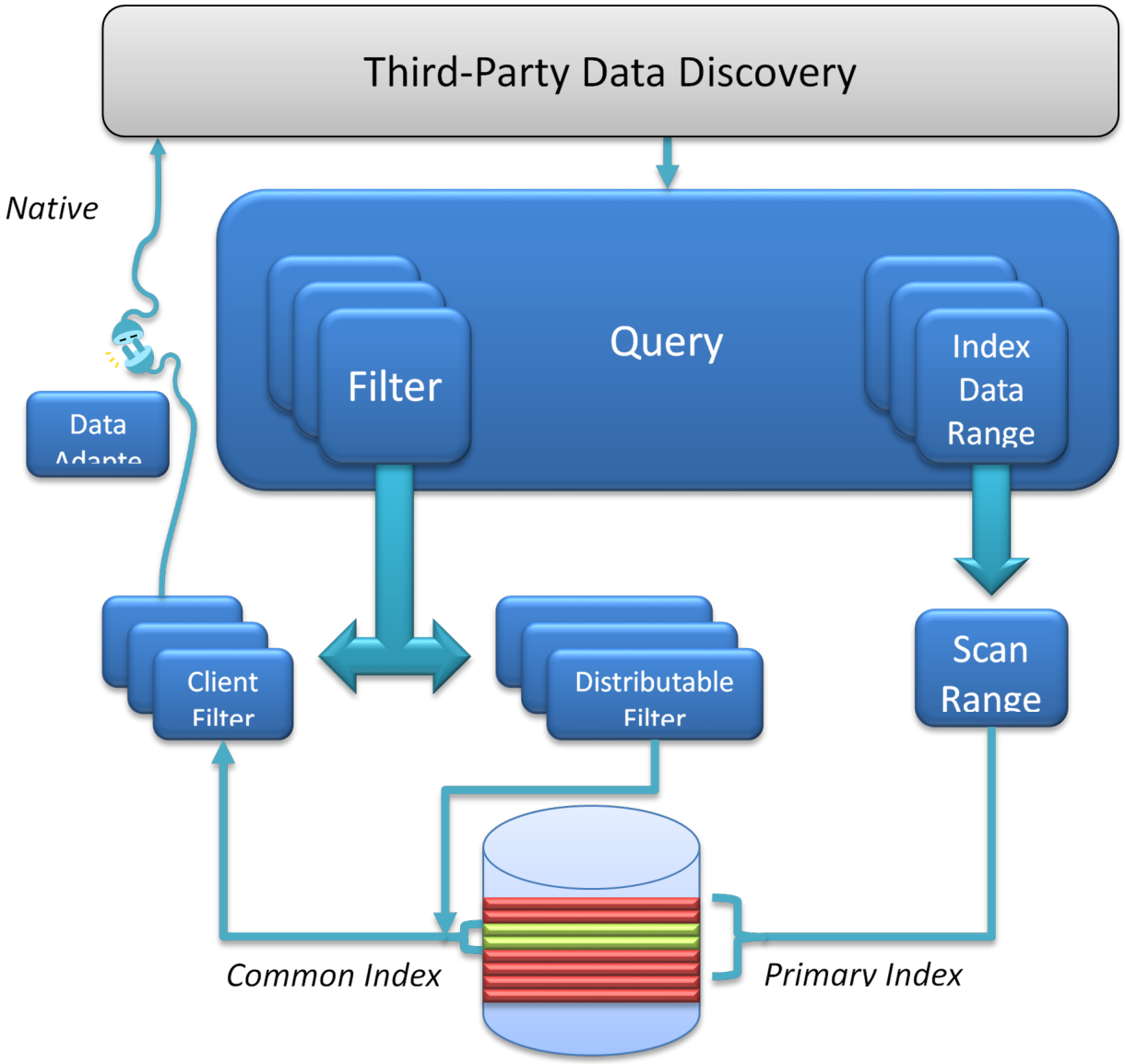
The above command will execute <algorithm> (such as dbscan), sourcing the data from the <store> datastore (see config addstore)

Analytic Commands

The full list of GeoWave analytic commands can be found in the GeoWave CLI Appendix (http://ngageoint.github.io/geowave/commands.html#analytic-commands).

Query

Overview



A query in GeoWave currently consists of a set of ranges on the dimensions of the primary index. Up to 3 dimensions (plus temporal optionally) can take advantage of any complex OGC geometry for the query window. For dimensions of 4 or greater the query can only be a set of ranges on each dimension (i.e. hyper-rectangle, etc.).

The query geometry is decomposed by GeoWave into a series of ranges on a one dimensional number line - based on a compact Hilbert space filling curve ordering. These ranges are sent through an Accumulo batch scanner to all the tablet servers. These ranges represent the coarse grain filtering.

At the same time the query geometry has been serialized and sent to custom Accumulo iterators. These iterators then do a second stage filtering on each feature for an exact intersection test. Only if the stored geometry and the query geometry intersect does the processing chain continue.

A second order filter can then be applied - this is used to remove features based on additional attributes - typically time or other feature attributes. These operators can only exclude items from the set defined by the range - they cannot include additional features. Think "AND" operators - not "OR".

A final filter is possible on the client set - after all the returned results have been aggregated together. Currently this is only used for final de-duplication. Whenever possible the distributed filter options should be used - as it splits the work load among all the tablet servers.

Third Party

GeoServer

Geowave supports both raster images and vector data exposed through Geoserver.

WFS-T

Geowave supports WFS-T for vector data by extending GeoTools. After following the deployment steps, Geowave appears as the data store types called 'GeoWave Datastore - accumulo' and 'GeoWave Datastore - hbase'.

On the Geowave Datastore - accumulo creation tab, the system prompts for the following connection parameters.

Name	Description	Constraints
zookeeper	Comma-separated list of Zookeeper host and port.	Host and port are separated by a colon (host:port).
instance	The Accumulo tablet server's instance name.	The name matches the one configured in Zookeeper.
user	The Accumulo user name.	The user should have administrative privileges to add and remove authorized visibility constraints.
password	Accumulo user's password.	
gwNamespace	The table namespace associated with this Accumlo data store	
Lock Management	Select one from a list of lock managers.	Zookeeper is required with a multiple Geoserver architecture.
Authorization Management Provider	Select from a list of providers.	
Authorization Data URL	The URL for an external supporting service or configuration file.	The interpretation of the URL depends on the selected provider.
Query Index Strategy	The pluggable query strategy to use for querying geowave tables - a reasonable default will be used if not supplied.	

On the Geowave Datastore - hbase creation tab, the system prompts for the following connection parameters.

Name	Description	Constraints
enableCustomFilters	Allows for the use of custom filters	Defaults to true
zookeeper	Comma-separated list of Zookeeper host and port.	Host and port are separated by a colon (host:port).
enableCoprocessors	Allows for the use of HBase co-processors	Defaults to true
gwNamespace	The table namespace associated with this Hbase data store	
verifyCoprocessors		Defaults to true

Name	Description	Constraints
Lock Management	Select one from a list of lock managers.	Zookeeper is required with a multiple Geoserver architecture.
Authorization Management Provider	Select from a list of providers.	
Authorization Data URL	The URL for an external supporting service or configuration file.	The interpretation of the URL depends on the selected provider.
Query Index Strategy	The pluggable query strategy to use for querying geowave tables - a reasonable default will be used if not supplied.	

Transactions

Transactions are initiated through a Transaction operation, containing inserts, updates and deletes to features. WFS-T supports feature locks across multiple requests by using a lock request followed by subsequent use of a provided lock ID. The Geowave implementation supports transaction isolation. Consistency during a commit is not fully supported. Thus, a failure during a commit of a transaction may leave the affected data in an intermediary state. Some deletions, updates or insertions may not be processed in such a case. The client application must implement its own compensation logic upon receiving a commit-time error response. As expected with Accumulo, operations on a single feature instances are atomic.

Inserted features are buffered prior to commit. The features are bulk fed to the data store when the buffer size is exceeded and when the transaction is committed. In support of atomicity and isolation, flushed features, prior to commit, are marked in a transient state, only visible to the controlling transaction. Upon commit, these features are 'unmarked'. The overhead incurred by this operation is avoided by increasing the buffer size to avoid pre-commit flushes.

Lock Management

Lock management supports life-limited locks on feature instances. There are only two supported lock managers: in-memory and Zookeeper. In-memory is suitable for single Geoserver instance installations.

Index Selection

Data written through WFS-T is indexed within a single index. The adapter inspects existing indices, finding one that matches the data requirements. A geo-temporal index is chosen for features with temporal attributes. The adapter creates a geospatial index upon failure of finding a suitable index. A geotemporal index is not created, regardless of the existence of temporal attributes. Currently, geotemporal indices lead to poor performance for queries requesting vectors over large spans of time.

Authorization Management

Authorization Management provides the set of credentials compared against the security labels attached to each cell. Authorization Management determines the set of authorizations associated with each WFS-T request. The available Authorization Management strategies are registered through the Server Provider model, within the file `META-INF/services/mil.nga.giat.geowave.vector.auth.AuthorizationFactorySPI`.

The provided implementations include the following

- Empty - Each request is processed without additional authorization.
- JSON - The requester user name, extracted from the Security Context, is used as a key to find the user's set of authorizations from a JSON file. The location of the JSON file is determined by the associated *Authorization Data URL* (e.g. file://opt/config/auth.json). An example of the contents of the JSON file is given below.

```JSON
{
  "authorizationSet": {
     "fred" : ["1","2","3"],
     "barney" : ["a"]
  }
}
```

Fred has three authorization labels. Barney has one.

Visibility Management

Visibility constraints, applied to feature instances during insertions, are ultimately determined in `mil.nga.giat.geowave.store.data.field.FieldWriter`, of which there are writers for each supported data type in Geoserver. By default, the set visibility expression attached to each feature property is empty. Visibility Management supports selection of a strategy by wrapping each writer to provide visibility. This alleviates the need to extend the type specific FieldWriters.

The visibility management strategy is registered through the Java Server Provider model, within in the file `META-INF/services/mil.nga.giat.geowave.vector.plugin.visibility.ColumnVisibilityManagement`. The only provided implementation is the `JsonDefinitionColumnVisibilityManagement`. The implementation expects an property within each feature instance to contain a JSON string describing how to set the visibility for each property of the feature instance. This approach allows each instance to determine its own visibility criteria.

Each name/value pair within the JSON structure defines the visibility for the associated feature property with the same name. In the following example, the *geometry* property is given a visibility `S`; the eventName is given a visibility `TS`.

```JSON
{ "geometry" : "S", "eventName": "TS" }
```

JSON attributes can be regular expressions, matching more than one feature property name. In the example, all properties except for those that start with 'geo' have visibility `TS`.

```JSON
{ "geo.*" : "S", ".*" : "TS" }
```

The order of the name/value pairs must be considered if one rule is more general than another, as shown in the example. The rule `.` **matches all properties. The more specific rule** `geo.` must be ordered first.

The system extracts the JSON visibility string from a feature instance property named `GEOWAVE_VISIBILITY`. Selection of an alternate property is achieved by setting the associated attribute descriptor 'visibility' to the boolean value TRUE.

Statistics

The adapter captures statistics for each numeric, temporal and geo-spatial attribute. Statistics are used to constrain queries and answer inquiries by GeoServer for data ranges, as required for map requests and calibration of zoom levels in OpenLayers.

Installation from RPM

Overview

There is a public GeoWave RPM Repo (http://ngageoint.github.io/geowave/packages.html) available with the following packages. As you'll need to coordinate a restart of Accumulo to pick up changes to the GeoWave iterator classes the repos default to be disabled so you can keep auto updates enabled. When ready to do an update simply add `--enablerepo=geowave` to your command. The packages are built for a number of different hadoop distributions (Cloudera, Hortonworks and Apache) the RPMs have the vendor name embedded as the second portion of the rpm name (geowave-apache-accumulo, geowave-hdp2-accumulo, geowave-cdh5-accumulo, geowave-apache-hbase, etc.)

Examples

BASH

```
# Use GeoWave repo RPM to configure a host and search for GeoWave RPMs to install
# Several of the rpms (accumulo, jetty and tools) are both GeoWave version and vendor version specific
# In the examples below the rpm name geowave-$VERSION-VENDOR_VERSION would be adjusted as needed
rpm -Uvh http://s3.amazonaws.com/geowave-rpms/release/noarch/geowave-repo-1.0-3.noarch.rpm
yum --enablerepo=geowave search geowave-0.9.3-apache

# Install GeoWave Accumulo iterator on a host (probably a namenode)
yum --enablerepo=geowave install geowave-0.9.3-apache-accumulo

# Update
yum --enablerepo=geowave install geowave-0.9.3-apache-*
```

GeoWave RPMs

Name	Description
geowave-*-accumulo	Accumulo Components
geowave-*-hbase	Hbase Components
geowave-*-core	Core (home directory and geowave user)
geowave-*-docs	Documentation (HTML, PDF and man pages)
geowave-*-tools	Command Line Tools (ingest, etc.)
geowave-*-jetty	GeoServer components installed into /usr/local/geowave/geoserver and available at http://FQDN:8000/geoserver/web
geowave-*-puppet	Puppet Scripts
geowave-*-single-host	All GeoWave Components installed on a single host (sometimes useful for development)
geowave-repo	GeoWave RPM Repo config file
geowave-repo-dev	GeoWave Development RPM Repo config file

RPM Installation Notes

RPM names contain the version in the name so support concurrent installations of multiple GeoWave and/or vendor versions. A versioned /usr/local/geowave-$GEOWAVE_VERSION-$VENDOR_VERSION directory is linked to /usr/local/geowave using alternatives ex: /usr/local/geowave → /usr/local/geowave-0.9.3-hdp2 but there could also be another /usr/local/geowave-0.9.2.1-cdh5 still installed but not the current default.

View geowave-home installed and default using alternatives

```
alternatives --display geowave-home
geowave-home - status is auto.
 link currently points to /usr/local/geowave-0.9.3-hdp2
/usr/local/geowave-0.9.3-hdp2 - priority 90
/usr/local/geowave-0.9.2.1-cdh5 - priority 89
Current `best' version is /usr/local/geowave-0.9.3-hdp2.
```

geowave-*-accumulo: This RPM will install the GeoWave Accumulo iterator into the local file system and then upload it into HDFS using the `hadoop fs -put` command. This means of deployment requires that the RPM is installed on a node that has the correct binaries and configuration in place to push files to HDFS, like your namenode. We also need to set the ownership and permissions correctly within HDFS and as such need to execute the script as a user that has superuser permissions in HDFS. This user varies by Hadoop distribution vendor. If the Accumulo RPM installation fails, check the install log located at `/usr/local/geowave/accumulo/geowave-to-hdfs.log` for errors. The script can be re-run manually if there was a problem that can be corrected like the HDFS service was not started. If a non-default user was used to install Hadoop you can specify a user that has permissions to upload with the --user argument `/usr/local/geowave/accumulo/deploy-to-geowave-to-hdfs.sh --user my-hadoop-user`

This only applies to the Accumulo RPM. There is no such requirement for the HBase RPM.

With the exception of the Accumulo RPM mentioned above there are no restrictions on where you install RPMs. You can install the rest of the RPMs all on a single node for development use or a mix of nodes depending on your cluster configuration.

Maven Repositories

Overview

There are public maven repositories available for both release and snapshot GeoWave artifacts (no transitive dependencies). Automated deployment is available, but requires a S3 access key (typically added to your ~/.m2/settings.xml)

Maven POM fragments

Releases

```xml
    <repository>
        <id>geowave-maven-releases</id>
        <name>GeoWave AWS Release Repository</name>
        <url>http://geowave-maven.s3-website-us-east-1.amazonaws.com/release</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
```

Snapshots

```xml
    <repository>
        <id>geowave-maven-snapshot</id>
        <name>GeoWave AWS Snapshot Repository</name>
        <url>http://geowave-maven.s3-website-us-east-1.amazonaws.com/snapshot</url>
        <releases>
            <enabled>false</enabled>
        </releases>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
```

Maven settings.xml fragments

(you probably don't need this unless you are deploying official GeoWave artifacts)

Snapshots

```xml
  <servers>
    <server>
      <id>geowave-maven-releases</id>
      <username>ACCESS_KEY_ID</username>
      <password>SECRET_ACCESS_KEY</password>
    </server>
    <server>
      <id>geowave-maven-snapshots</id>
      <username>ACCESS_KEY_ID</username>
      <password>SECRET_ACCESS_KEY</password>
    </server>
  </servers>
```

Installation from Source

GeoServer

GeoServer Versions

GeoWave has to be built against specific versions of GeoWave and GeoTools. To see the currently supported versions look at the build matrix section of the .travis.yml file in the root directory of the project. All of the examples below use the variable $BUILD_ARGS to represent your choice of all the dependency versions.

Example build args:

```
export BUILD_ARGS="-Daccumulo.version=1.7.2 -Daccumulo.api=1.7 -Dhbase.version=1.3.0 -Dhadoop.version=2.7.3 -Dgeotools.version=16.0 -Dgeoserver.version=2.10.0" (1)
```

(1) Examples of current build args can be seen in the top level .travis.yml file in the env/matrix section

GeoServer Install

First we need to build the GeoServer plugin - from the GeoWave root directory:

```
mvn package -P geotools-container-singlejar $BUILD_ARGS (1)
```

(1) You can speed up the build by skipping tests by adding -Dfindbugs.skip=true -Dformatter.skip=true -DskipITs=true -DskipTests=true

let's assume you have GeoServer deployed in a Tomcat container in /opt/tomcat

```
cp deploy/target/*-geoserver-singlejar.jar /opt/tomcat/webapps/geoserver/WEB-INF/lib/
```

and re-start Tomcat

If you used the RPMs to build and install GeoWave then GeoSever will have already been deployed via Jetty. The jar file is installed in /usr/local/geowave/geoserver/webapps/geoserver/WEB-INF/lib/

You can restart Jetty by restarting the GeoWave service.

Accumulo

Accumulo Versions

GeoWave has been tested and works against accumulo 1.5.0 through 1.7.2 . Ensure you've set the desired version in the BUILD_ARGS environment variable

Accumulo Install

```
mvn package -P accumulo-container-singlejar $BUILD_ARGS (1)
```

(1) You can speed up the build by skipping tests by adding -Dfindbugs.skip=true -Dformatter.skip=true -DskipITs=true -DskipTests=true

---

Running from EMR

This is meant to be a high level overview for those already familiar with AWS EMR. For a step by step walkthrough of setting up GeoWave on an EMR cluster, please see our Quickstart Guide (http://ngageoint.github.io/geowave/quickstart.html).

Provisioning

The configuration files needed to use GeoWave from EMR are automatically generated and stored in an s3 bucket at s3.amazonaws.com/geowave/latest/scripts/emr/accumulo/ and s3.amazonaws.com/geowave/latest/scripts/emr/hbase/. EMR expects that all config files are available from S3 so the first step would be to create an S3 bucket and then copy the bootstrap-geowave.sh script into your own S3 bucket adjusting the path used in the command as needed. Alternatively, you can just reference the bootstrap-geowave.sh script in your command without transferring it into a separate bucket.

The command below is an example of using the API to launch an EMR cluster, you could also provide this same information from the console. There are a number of fields that are unique to each deployment most of which you'll see with a placeholder like YOUR_KEYNAME in the command below. If running from a bash script you can use variable replacement to collect and substitute value such as the number of worker instances. Use the command below as a starting point, it will not work if you try to cut and paste.

Once the process of cluster initialization has started you will see the cluster appear in the EMR console immediately. The GeoWave portion of the process does not occur until the Hadoop and Spark portions of the initializations have completed which takes approximately 4-5 minutes. Once the GeoWave components have been installed there is an optional volume initialization step that will read every block of each volume to clear any initialization flags that may have been set. This option should be used when you want to benchmark an operation but can probably be skipped if you're primarily interested in quickly setting up a cluster to test some capability.

```bash
aws emr create-cluster \
  --name "geowave-emr" \
  --instance-groups InstanceGroupType=MASTER,InstanceCount=1,InstanceType=m3.xlarge
InstanceGroupType=CORE,InstanceCount=${NUM_WORKERS},InstanceType=m3.xlarge \
  --ec2-attributes "KeyName=${YOUR_KEYNAME},SubnetId=${YOUR_SUBNET_ID}" \
  --region ${YOUR_REGION} \
  --release-label emr-5.1.0 \
  --applications Name=Hadoop\
  --use-default-roles \
  --no-auto-terminate \
  --bootstrap-actions Path=s3://${BOOTSTRAP_SCRIPT_DIR}/bootstrap-geowave.sh,Name=Bootstrap_GeoWave_Node \
  --tags "Name=geowave-emr-worker"
```

Connecting

To connect to the cluster you'd use ssh to connect to the console and another ssh connection setting up a SOCKS proxy to connect via a web browser to the various web consoles. The key you'd use in both cases would be the one you specified in the ec2-attributes KeyName portion of the command.

- Example SSH Console connection: `ssh -i YOUR_KEYFILE ec2-user@MASTER_FQDN`
- Example SOCKS Proxy connection: `ssh -i YOUR_KEYFILE -ND 5556 ec2-user@MASTER_FQDN`

After establishing the SOCKS proxy you'd then configure your browser to use the port you specified. A more detailed explanation can be found in the AWS docs here (https://docs.aws.amazon.com/ElasticMapReduce/latest/ManagementGuide/emr-ssh-tunnel.html).

Links

After setting up a SOCKS proxy to the master node you should be able to connect to any of the following web consoles hosted from the cluster. The name of the master node can be found in the description of the EMR job in the AWS console. Example: Master public DNS: ec2-52-91-31-196.compute-1.amazonaws.com (This is refered to as the **MASTER_FQDN** in the links below)

- Accumulo: http://${MASTER_FQDN}:50095
- Ganglia Monitoring: http://${MASTER_FQDN}/ganglia/

- GeoWave GeoServer: http://${MASTER_FQDN}:8000/geoserver/web
- HDFS: http://${MASTER_FQDN}:50070
- HUE: http://${MASTER_FQDN}:8888
- YARN: http://${MASTER_FQDN}:8088

Configuration

After the cluster has finished initializing you should be able to ssh into the master node and perform the final bits of project specific GeoWave configuration. The root password for Accumulo is set at the top of the bootstrap-geowave.sh script. You'd want to log into Accumulo and perform steps listed in the Accumulo Configuration section of the documentation (Not needed for HBase). The latest iterator built for Apache Hadoop will have been uploaded into HDFS but no user accounts, namespaces or VFS contexts will have been configured. All of these are described with examples in both the GeoWave and Accumulo documentation.

Accumulo Configuration

Overview

The two high level tasks to configure Accumulo for use with GeoWave are to ensure the memory allocations for the master and tablet server processes are adequate and to add the GeoWave Accumulo iterator to a classloader. The iterator is a rather large file so ensure the Accumulo Master process has at least 512m of heap space and the Tablet Server processes have at least 1g of heap space.

The recommended Accumulo configuration for GeoWave requires several manual configuration steps but isolates the GeoWave libraries in application specific classpath(s) reducing the possibility of dependency conflict issues. A single user for all of geowave data or a user per data type are two of the many local configuration options just ensure each namespace containing GeoWave tables is configured to pick up the geowave-accumulo.jar.

Procedure

1. Create a user and namespace
2. Grant the user ownership permissions on all tables created within the application namespace
3. Create an application or data set specific classpath
4. Configure all tables within the namespace to use the application classpath

```BASH
accumulo shell -u root
createuser geowave (1)
createnamespace geowave
grant NameSpace.CREATE_TABLE -ns geowave -u geowave (2)
config -s general.vfs.context.classpath.geowave=hdfs://${MASTER_FQDN}:8020/${ACCUMULO_ROOT}/lib/[^.].*.jar (3)
config -ns geowave -s table.classpath.context=geowave (4)
exit
```

(1) You'll be prompted for a password

(2) Ensure the user has ownership of all tables created within the namespace

(3) The Accumulo root path in HDFS varies between hadoop vendors. For Apache and Cloudera it is /accumulo and for Hortonworks it is /apps/accumulo

(4) Link the namespace with the application classpath, adjust the labels as needed if you've used different user or application names

These manual configuration steps have to be performed before attempting to create GeoWave index tables. After the initial configuration you may elect to do further user and namespace creation and configuring to provide isolation between groups and data sets.

Managing

After installing a number of different iterators you may want to figure out which iterators have been configured.

```BASH
# Print all configuration and grep for line containing vfs.context configuration and also show the following line
accumulo shell -u root -p ROOT_PWD -e "config -np" | grep -A 1 general.vfs.context.classpath
```

You will get back a listing of context classpath override configurations which map the application or user context you configured to a specific iterator jar in HDFS.

Versioning

It's of critical importance to ensure that the various GeoWave components are all the same version and that your client is of the same version that was used to write the data.

Basic

The RPM packaged version of GeoWave puts a timestamp in the name so it's pretty easy to verify that you have a matched set of RPMs installed. After an update of the components you must restart Accumulo to get vfs to download the new versions and this should keep everything synched.

Compare version and timestamps of installed RPMs

```
[spohnae@c1-master ~]$ rpm -qa | grep geowave
geowave-0.9.3-apache-core-0.9.3-201602012009.noarch
geowave-0.9.3-apache-jetty-0.9.3-201602012009.noarch
geowave-0.9.3-apache-accumulo-0.9.3-201602012009.noarch
geowave-0.9.3-apache-tools-0.9.3-201602012009.noarch
```

BASH

Advanced

When GeoWave tables are first accessed on a tablet server the vfs classpath tells Accumulo where to download the jar file from HDFS. The jar file is copied into the local /tmp directory (the default general.vfs.cache.dir setting anyway) and loaded onto the classpath. If there is ever doubt as to if these versions match you can use the commands below from a tablet server node to verify the version of this artifact.

Commit hash of the jar in HDFS

```
sudo -u hdfs hadoop fs -cat /accumulo/classpath/geowave/geowave-accumulo-build.properties | grep scm.revision | sed
s/project.scm.revision=(1)
```

BASH

(1) The root directory of Accumulo in various distributions can vary, check with hadoop fs -ls / first to ensure you have the correct initial path

Compare with the versions downloaded locally

```
sudo find /tmp -name "*geowave-accumulo.jar" -exec unzip -p {} build.properties  \; | grep scm.revision | sed
s/project.scm.revision=//
```

BASH

Example

```
[spohnae@c1-node-03 ~]$ sudo -u hdfs hadoop fs -cat /${ACCUMULO_ROOT}/lib/geowave-accumulo-build.properties | grep
scm.revision | sed s/project.scm.revision=//
294ffb267e6691de3b9edc80e312bf5af7b2d23f (1)
[spohnae@c1-node-03 ~]$ sudo find /tmp -name "*geowave-accumulo.jar" -exec unzip -p {} build.properties  \; | grep
scm.revision | sed s/project.scm.revision=//
294ffb267e6691de3b9edc80e312bf5af7b2d23f (2)
294ffb267e6691de3b9edc80e312bf5af7b2d23f (2)
25cf0f895bd0318ce4071a4680d6dd85e0b34f6b
```

BASH

(1) This is the version loaded into hdfs and should be present on all tablet servers once Accumulo has been restarted

(2) The find command will probably locate a number of different versions depending on how often you clean out /tmp.

There may be multiple versions copies present, one per JVM, the error scenario is when a tablet server is missing the correct iterator jar.

Building

GeoWave will shortly be available in maven central (for tagged releases), but until then - or to get the latest features - building GeoWave from source is the best bet.

Application Dependencies

This *ultra* quickstart assumes you have installed and configured:

- Git (http://git-scm.com/)
- Java JDK (http://www.oracle.com/technetwork/java/javase/downloads/index.html) (>= 1.8). The OracleJDK is the most thoroughly tested, but there are no known issues with OpenJDK.
- Maven (https://maven.apache.org/) >= 3.2.1
- GeoServer (http://geoserver.org/) instance >= 2.5.2
- Apache Accumulo (https://accumulo.apache.org/) version 1.5 or greater is required. 1.5.0, 1.5.1, and 1.6.0 have all been tested.
- Apache HBase (https://hbase.apache.org/) >= 1.2.1
- Apache Hadoop (http://hadoop.apache.org/) versions 1.x and 2.x
- Cloudera (http://cloudera.com/content/cloudera/en/home.html) CDH4 and CDH5
- Hortonworks Data Platform (http://hortonworks.com/hdp/) 2.1+
- Java Advanced Imaging (http://www.oracle.com/technetwork/articles/javaee/jai-142803.html) and Java Image I/O (https://docs.oracle.com/javase/8/docs/technotes/guides/imageio/) are also both required to be installed on the GeoServer instance(s) *as well* as on the Accumulo nodes. The Accumulo support is only required for certain functions (distributed rendering) - so this may be skipped in some cases.

Maven dependencies

Required repositories not in Maven Central have been added to the parent POM. Specifically the cloudera and opengeo repos.

Build Process

Checkout GeoWave, set your preferred dependencies as build arguments and then run maven install.

```
export BUILD_ARGS="-Daccumulo.version=1.7.2 -Daccumulo.api=1.7 -Dhbase.version=1.3.0 -Dhadoop.version=2.7.3 -
Dgeotools.version=16.0 -Dgeoserver.version=2.10.0" (1)
git clone https://github.com/ngageoint/geowave.git (2)
cd geowave
mvn install $BUILD_ARGS (3)
```

BASH

1. Examples of current build args can be seen in the top level .travis.yml file in the env/matrix section
2. If you don't need the complete history and want to speed up the clone you can limit the depth of your checkout with --depth NUM_COMMITS
3. You can speed up the build by skipping tests by adding -Dfindbugs.skip=true -Dformatter.skip=true -DskipITs=true -DskipTests=true

Integration Tests: Windows

Integration tests are currently not working on Windows out of the box. If you install cygwin and set the environmental variable CYGPATH to the location of the cygpath binary provided by cygwin then this should work.

Docker Build Process

We have support for building both the GeoWave jar artifacts and RPMs from Docker containers. This capability is useful for a number of different situations:

- Jenkins build workers can run Docker on a variety of host operating systems and build for others

- Anyone running Docker will be able to duplicate our build and packaging environments
- Will allow us to build on existing container clusters instead of single purpose build VMs

If building artifacts using Docker containers interests you check out the README in `deploy/packaging/docker`

Jace JNI Proxies

Using Jace, we are able to create JNI proxy classes for GeoWave which can be used in C/C++ applications.

Boost is required when using the Jace bindings.

Prepackaged Source and Binaries

There is a public GeoWave RPM Repo (http://ngageoint.github.io/geowave/packages.html) where you can download a tarball for the GeoWave Jace bindings for your desired platform. If your platform is not available, there is a source tarball which can be used in conjunction with CMake to build the GeoWave Jace bindings for your desired platform.

Generate Proxies and Build from Source

If you want, you can generate and build the Jace proxies yourself.

Step 1 - Checkout Jace and GeoWave

First, we need to clone Jace and GeoWave.

```
$ git clone git@github.com:jwomeara/jace.git
$ git clone git@github.com:ngageoint/geowave.git
```

Note: We are using a non-standard Jace implementation.

Step 2 - Install Jace

First, we need to install Jace v1.3.0. This is the software which is used to generate the C++ proxy classes.

```
$ cd jace
$ git checkout tags/v1.3.0
$ mvn clean install -Dsources
```

Step 3 - Generate GeoWave Jace Proxies

Here, we will specify a Maven profile which specifies that we are building jace proxies.

```
$ cd geowave
$ mvn clean package -pl deploy -am -P generate-geowave-jace -DskipTests
```

This generates the source and header files required to build GeoWave. To build the library, simply run cmake, followed by make.

Note: To build static libraries use -DBUILD_SHARED_LIBS=OFF, otherwise use -DBUILD_SHARED_LIBS=ON

Mapnik Plugin Configuration

Mapnik

Mapnik (http://mapnik.org/) is an open source toolkit for developing mapping applications. GeoWave is supported as a plugin for Mapnik for reading vector data from Accumulo.

PDAL Plugin Configuration

PDAL

The Point Data Abstraction Library PDAL (http://www.pdal.io/index.html) is a BSD licensed library for translating and manipulating point cloud data of various formats. GeoWave is supported as a plugin for PDAL for both reading and writing data to Accumulo.

Note: These instructions assume that you are using prepackaged binaries.

Configure CMake for PDAL

To configure PDAL to run with GeoWave, there are a few CMake options which need to be configured. While some of the options (namely the JAVA options) may configure automatically, some will need to be set manually. Refer to the table below to get an idea for how these options would be configured on Ubuntu 14.04 LTS.

Option	Value	Automatically Configured?
BUILD_PLUGIN_GEOWAVE	ON	
BUILD_GEOWAVE_TESTS	ON	
GEOWAVE_RUNTIME_JAR	/path/to/geowave/geowave-runtime.jar	
GEOWAVE_INCLUDE_DIR	/path/to/geowave/include	
GEOWAVE_LIBRARY	/path/to/geowave/libgeowave.so	
JAVA_AWT_INCLUDE_PATH	/usr/lib/jvm/java-8-oracle/include	X
JAVA_INCLUDE_PATH	/usr/lib/jvm/java-8-oracle/include	X
JAVA_INCLUDE_PATH2	/usr/lib/jvm/java-8-oracle/include/linux	X
JAVA_AWT_LIBRARY	/usr/lib/jvm/java-8-oracle/jre/lib/amd64/libjawt.so	X
JAVA_JVM_LIBRARY	/usr/lib/jvm/java-8-oracle/jre/lib/amd64/server/libjvm.so	X

Note: As Boost is a PDAL dependency, it should already be included.

Build PDAL

Once CMake is configured, you can proceed with your normal PDAL build process.

Last, but not least, when building shared libraries you should ensure that the libraries specified above are available via PATH or LD_LIBRARY_PATH.

Within the PDAL documentation, you can see examples of how GeoWave can be used as both a reader (http://www.pdal.io/stages/readers.geowave.html) and writer (http://www.pdal.io/stages/writers.geowave.html).

Puppet

Overview

A GeoWave Puppet module (http://puppetlabs.com/) has been provided as part of both the tar.gz archive bundle and as an RPM. This module can be used to install the various GeoWave services onto separate nodes in a cluster or all onto a single node for development.

There are a couple of different RPM repo settings that may need setting. As the repo is disabled by default to avoid picking up new Accumulo iterator jars without coordinating a service restart you'll have to enable and then disable in consecutive Puppet runs to do the initial install.

Options

geowave_version

The desired version of GeoWave to install, ex: '0.9.3'. We support concurrent installs but only one will be active at a time.

hadoop_vendor_version

The Hadoop framework vendor and version against which GeoWave was built. Examples would be cdh5 or hdp2, check the available packages (http://ngageoint.github.io/geowave/packages.html) site for currently supported hadoop distributions.

install_accumulo

Install the GeoWave Accumulo Iterator on this node and upload it into HDFS. This node must have a working HDFS client.

install_app

Install the GeoWave ingest utility on this node. This node must have a working HDFS client.

install_app_server

Install Jetty with Geoserver and GeoWave plugin on this node.

http_port

The port on which the Jetty application server will run - defaults to 8080.

repo_base_url

Used with the optional geowave::repo class to point the local package management system at a source for GeoWave RPMs. The default location is http://s3.amazonaws.com/geowave-rpms/release/noarch/

repo_enabled

To pick up an updated Accumulo iterator you'll need to restart the Accumulo service. As we don't want to pick up new RPMs with something like a yum-cron job without coordinating a restart so the repo is disabled by default.

repo_refresh_md

The number of seconds before checking for new RPMs. On a production system the default of every 6 hours should be sufficient but you can lower this down to 0 for a development system on which you wish to pick up new packages as soon as they are made available.

Examples

Development

Install everything on a one node development system, use the GeoWave Development RPM Repo and force a check for new RPMs with every pull (don't use cached metadata)

```ruby
# Dev VM
class { 'geowave::repo':
  repo_enabled    => 1,
  repo_refresh_md => 0,
} ->
class { 'geowave':
  geowave_version       => '0.9.3',
  hadoop_vendor_version => 'apache',
  install_accumulo      => true,
  install_app           => true,
  install_app_server    => true,
}
```

Clustered

Run the application server on a different node, use a locally maintained rpm repo vs. the one available on the Internet and run the app server on an alternate port so as not to conflict with another service running on that host.

```ruby
# Master Node
node 'c1-master' {
  class { 'geowave::repo':
    repo_base_url   => 'http://my-local-rpm-repo/geowave-rpms/dev/noarch/',
    repo_enabled    => 1,
  } ->
  class { 'geowave':
    geowave_version       => '0.9.3',
    hadoop_vendor_version => 'apache',
    install_accumulo      => true,
    install_app           => true,
  }
}

# App server node
node 'c1-app-01' {
  class { 'geowave::repo':
    repo_base_url   => 'http://my-local-rpm-repo/geowave-rpms/dev/noarch/',
    repo_enabled    => 1,
  } ->
  class { 'geowave':
    geowave_version       => '0.9.3',
    hadoop_vendor_version => 'apache',
    install_app_server    => true,
    http_port             => '8888',
  }
}
```

Puppet script management

As mentioned in the overview the scripts are available from within the GeoWave source tar bundle (http://ngageoint.github.io/geowave/packages.html) (Search for gz to filter the list) or you could use the RPM package to install and pick up future updates on your puppet server.

Source Archive

Unzip the source archive, locate puppet-scripts.tar.gz and manage the scripts yourself on your Puppet Server

RPM

There's a bit of a boostrap issue when first configuring the Puppet server to use the geowave puppet RPM as yum won't know about the rpm repo and the GeoWave Repo Puppet class hasn't been installed yet. There is an RPM available that will set up the yum repo config after which you should install geowave-puppet manually and proceed to configure GeoWave on the rest of the cluster using Puppet.

```bash
rpm -Uvh http://s3.amazonaws.com/geowave-rpms/release/noarch/geowave-repo-1.0-3.noarch.rpm
yum --enablerepo=geowave install geowave-puppet
```

How to Contribute

GeoWave is an open source project and we welcome contributions from the community.

Pull Requests

All pull request contributions to this project will be released under the Apache 2.0 license.

Software source code previously released under an open source license and then modified by NGA staff is considered a "joint work" (see *17 USC 101*); it is partially copyrighted, partially public domain, and as a whole is protected by the copyrights of the non-government authors and must be released according to the terms of the original open source license.

Documentation

Overview

The documentation is writen in AsciiDoc (http://www.methods.co.nz/asciidoc/index.html) which is a plain-text markup format that can be created using any text editor and read "as-is", or rendered to several other formats like HTML, PDF or EPUB.

Helpful Links:

- What is Asciidoc? (http://asciidoctor.org/docs/what-is-asciidoc/)
- Writer's Guide (http://asciidoctor.org/docs/asciidoc-writers-guide/)
- AsciiDoc Syntax Reference (http://asciidoctor.org/docs/asciidoc-syntax-quick-reference/)

Ordering

All of the content stored in the `docs/content` directory of this project will be rendered into a single web page with an auto-generated table of contents and a PDF. The order in which the pages appear is determined by the sort order of the file names given to the ASCIIDOC files in the `docs/content` directory so a numeric prefix has been given to each file. Gaps can be left in between the numbers (only the sort order is important) to allow for future edits without having to renumber other documents that will appear after the new content.

Preview

To preview markup as HTML before making a commit there are plugins available various text editors and IDEs that can be used while editing. If your preferred text editor has no plugin available there's a Firefox AsciiDoc Plugin (https://github.com/asciidoctor/asciidoctor-firefox-addon) available which allows for previewing with a quick refresh of the browser.

Transformation

To preview the entire finished web page or see the generated PDF you'll need to run the transformation process.

Generate Documentation

```
cd geowave
mvn -P html install (1) (2)
```

1. To generate the documentation in pdf form use the pdf profile instead of the html profile.
2. You can speed up the build by skipping tests by adding -Dfindbugs.skip=true -Dformatter.skip=true -DskipITs=true -DskipTests=true

The source documents will be transformed and will be available for inspection in the `geowave/target/site/` directory.

Javadocs

Generate Javadocs

```
cd geowave
mvn -q javadoc:aggregate
```

The Javadocs will be available for inspection in the `geowave/target/site/apidocs` directory.

Appendices

Version

This documentation was generated for GeoWave version 0.9.4 from commit ffc45cd0b829bd44a20edee4d57629fee77ec55a.

Topics in need of documentation

- Ingest Examples
- Query Iterators
- Query CQL
- MapReduce Input Format
- Analytics - Kernel Density
- Analytics - K-Means Clustering
- Examples - Shapefile
- Examples - OSM GPX Data
- Examples - OSM Feature Data
- Index Tuning
- Supported Types
- Editing from GitHub

Last updated 2017-05-02 15:00:38 +00:00