Using Druid and Apache Hive

Date of Publish: 2018-07-12
Contents

Accelerating Hive queries using Druid

How Druid indexes Hive data

Transform Apache Hive Data to Druid

Anatomy of a Hive-to-Druid data transformation

Druid and Hive tuning
Accelerating Hive queries using Druid

You can perform interactive analytic queries on real-time and historical data using the HDP integration of Hive and Apache Druid (incubating). You can discover existing Druid data sources as external tables, create or ingest batch data into Druid, set up Druid-Kafka streaming ingestion using Hive, and query Druid data sources from Hive.

The integration of Hive with Druid places a SQL layer on Druid. After Druid ingests data from a Hive enterprise data warehouse (EDW), the interactive and sub-second query capabilities of Druid can be used to accelerate queries on historical data from the EDW. Hive integration with Druid enables applications such as Tableau to scale while queries run concurrently on both real-time and historical data. The following figure is an overview of how Hive historical data can be brought into a Druid environment. Queries analyzing Hive-sourced data are run directly on the historical nodes of Druid after indexing between the two databases completes.

Related Information
Druid Integration

How Druid indexes Hive data

Before you can create a Druid data source based on Hive data, you must understand how Hive external table data maps to the column orientation and segment files of Druid.

Mapping of a Hive external table to a Druid file

Each Druid segment consists of the following objects to facilitate fast lookup and aggregation:

**Timestamp column**

The SQL-based timestamp column is filled in based on how you set the time granularity of imported Hive data and what time range of data is selected in the Hive external table. This column is essential for indexing the data in Druid because Druid itself is a time-series database. The timestamp column must be named `__time`.

**Dimension columns**

The dimension columns are used to set string attributes for search and filter operations. To index a Hive-sourced column as a Druid dimension column, you must cast the column as a string type.

**Metric columns**

Metric columns are used to index metrics for use as aggregates or measures. To index a Hive-sourced column as a Druid metric column, you must cast the column as a Hive numeric data type.
The following figure shows how you can categorize Druid data into three types of columns.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Page</th>
<th>Username</th>
<th>Gender</th>
<th>City</th>
<th>Characters Added</th>
<th>Characters Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011-01-01T00:00:00Z</td>
<td>Justin Bieber</td>
<td>Boxer</td>
<td>Male</td>
<td>San Francisco</td>
<td>1800</td>
<td>25</td>
</tr>
<tr>
<td>2011-01-01T00:00:00Z</td>
<td>Justin Bieber</td>
<td>Reach</td>
<td>Male</td>
<td>Waterloo</td>
<td>2912</td>
<td>42</td>
</tr>
<tr>
<td>2011-01-01T02:00:00Z</td>
<td>Kesha</td>
<td>Hetz</td>
<td>Male</td>
<td>Calgary</td>
<td>1953</td>
<td>7</td>
</tr>
<tr>
<td>2011-01-01T02:00:00Z</td>
<td>Kesha</td>
<td>Xeno</td>
<td>Male</td>
<td>Taoyuan</td>
<td>3194</td>
<td>170</td>
</tr>
</tbody>
</table>

Related Information
Druid Integration

Transform Apache Hive Data to Druid

You can execute a Hive query to transform data in Hive to a data source in Druid.

Before you begin

- If you use Kerberos, configure and run Hive low-latency analytical processing (LLAP).
- You set up a table, ssb_10_flat_orc as defined in the Star Schema Benchmark example.

About this task

A Hive SQL command, an excerpt from the Star Schema Benchmark using the Hive/Druid Integration, invokes the Druid storage handler, specifies Druid segment granularity, and maps selected Hive columns to Druid column types.

Procedure

1. Put all the Hive data to undergo ETL in a Hive table.
2. Execute a Hive query to set the location of the Druid broker using a DNS name or IP address and port 8082, the default broker text listening port.
   ```
   SET hive.druid.broker.address.default=10.10.20.30:8082;
   ```

   If you installed the Hive and Druid services using Ambari, you can skip this step.

3. Run a CREATE TABLE AS SELECT statement to create a new Druid datasource from the Hive table you selected in step 1.
   The following example pushes Hive data to Druid.
   ```
   CREATE TABLE ssb_druid_hive
   STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
   TBLPROPERTIES ( 
   "druid.segment.granularity" = "MONTH",
   "druid.query.granularity" = "DAY")
   AS
   SELECT
   cast(d_year || '-' || d_monthnuminyear || '-' || d_daynuminmonth as timestamp) as __time,
   cast(c_city as string) c_city,
   cast(c_nation as string) c_nation,
   cast(c_region as string) c_region,
   cast(d_weeknuminyear as string) d_weeknuminyear,
   cast(d_year as string) d_year,
   cast(d_yearmonth as string) d_yearmonth,
   cast(d_yearmonthnum as string) d_yearmonthnum,
   cast(lo_discount as string) lo_discount,
   cast(lo_quantity as string) lo_quantity,
   ```
Anatomy of a Hive-to-Druid data transformation

A breakdown of the example SQL code that transforms Hive data into a Druid datasource helps you understand how to transform your own data.

SQL example code

The following SQL statement contains the main elements of a statement that can transform Hive data into a time series-based Druid datasource. You need to replace the values in the statement to match your data warehouse environment and analytics parameters.

```
CREATE TABLE ssb_druid_hive
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES (
"druid.segment.granularity" = "MONTH",
"druid.query.granularity" = "DAY")
AS
SELECT
  cast(d_year || '-' || d_monthnuminyear || '-' || d_daynuminmonth as timestamp) as __time,
  cast(c_city as string) c_city,
  cast(c_nation as string) c_nation,
  cast(c_region as string) c_region,
  cast(d_weeknuminyear as string) d_weeknuminyear,
  cast(d_year as string) d_year,
  cast(d_yearmonth as string) d_yearmonth,
  cast(d_yearmonthnum as string) d_yearmonthnum,
  cast(lo_discount as string) lo_discount,
  cast(lo_quantity as string) lo_quantity,
  cast(p_brand1 as string) p_brand1,
  cast(p_category as string) p_category,
  cast(p_mfgr as string) p_mfgr,
  cast(s_city as string) s_city,
  cast(s_nation as string) s_nation,
  cast(s_region as string) s_region,
  lo_revenue,
  lo_extendedprice * lo_discount discounted_price,
  lo_revenue - lo_supplycost net_revenue
FROM
  ssb_10_flat_orc.customer, ssb_10_flat_orc.dates,
  ssb_10_flat_orc.lineorder,
  ssb_10_flat_orc.part, ssb_10_flat_orc.supplier
where
  lo_orderdate = d_datekey and lo_partkey = p_partkey
and lo_suppkey = s_suppkey and lo_custkey = c_custkey;
```
FROM ssb_10_flat_orc.customer, ssb_10_flat_orc.dates, ssb_10_flat_orc.lineorder, ssb_10_flat_orc.part, ssb_10_flat_orc.supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey
and lo_suppkey = s_suppkey and lo_custkey = c_custkey;

Explanation of SQL example

The following breakdown of the preceding SQL statement explains the main elements of a statement that can transform Hive data into a time series-based Druid datasource. You need to replace the values in the statement to match your data warehouse environment and analytics parameters.

CREATE TABLE ssb_druid_hive

STORED BY, 'org.apache.hadoop.hive., druid.DruidStorageHandler'

TBLPROPERTIES ("druid.segment.granularity" = "MONTH", "druid.query.granularity" = "DAY")

AS SELECT cast(d_year || '-' || d_monthnuminyear || '-' || d_daynuminmonth as timestamp) as `__time`,
cast(c_city as string) c_city, cast(c_nation as string) c_nation, cast(c_region as string) c_region,
cast(d_weeknuminyear as string) d_weeknuminyear, ...

lo_extendedprice * lo_discount discounted_price,
lo_revenue - lo_supplycost net_revenue

FROM ssb_10_flat_orc.customer,
ssb_10_flat_orc.dates, ...

Table Property Required Description Valid Values

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>druid.segment.popularity</td>
<td>No</td>
<td>Defines how the data is physically partitioned. The values that are permissible here correspond with Druid segment granularity.</td>
<td>&quot;YEAR&quot;, &quot;MONTH&quot;, &quot;WEEK&quot;, &quot;DAY&quot;, &quot;HOUR&quot;, &quot;MINUTE&quot;, &quot;SECOND&quot;</td>
</tr>
<tr>
<td>druid.query.granularity</td>
<td>No</td>
<td>Defines how much granularity to store in a segment. The values that are permissible here correspond with Druid query granularity.</td>
<td>&quot;YEAR&quot;, &quot;MONTH&quot;, &quot;WEEK&quot;, &quot;DAY&quot;, &quot;HOUR&quot;, &quot;MINUTE&quot;, &quot;SECOND&quot;</td>
</tr>
</tbody>
</table>

If you need Druid to ingest Hive data that follows the same schema as the first data set that you transformed, you can do so with the INSERT INTO statement.

Druid and Hive tuning

As administrator, you can set druid.hive properties to improve Druid-Hive performance.
Performance related druid.hive properties

If Hive and Druid are installed with Ambari, the properties are set and tuned for your cluster automatically. However, you can fine-tune some properties if you detect performance problems with applications that are running the queries. The following list includes some of the Druid properties that can be used by Hive. As an HDP administrator, you can troubleshoot and customize a Hive-Druid integration using these properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hive.druid.indexer.segments.granularity</td>
<td>Granularity of the segments created by the Druid storage handler.</td>
</tr>
<tr>
<td>hive.druid.indexer.partition.size.max</td>
<td>Maximum number of records per segment partition.</td>
</tr>
<tr>
<td>hive.druid.indexer.memory.rownum.max</td>
<td>Maximum number of records in memory while storing data in Druid.</td>
</tr>
<tr>
<td>hive.druid.broker.address.default</td>
<td>Address of the Druid broker node. When Hive queries Druid, this address must be declared.</td>
</tr>
<tr>
<td>hive.druid.coordinator.address.default</td>
<td>Address of the Druid coordinator node. It is used to check the load status of newly created segments.</td>
</tr>
<tr>
<td>hive.druid.select.threshold</td>
<td>When a SELECT query is split, this is the maximum number of rows that Druid attempts to retrieve.</td>
</tr>
<tr>
<td>hive.druid.http.numConnection</td>
<td>Number of connections used by the HTTP client.</td>
</tr>
<tr>
<td>hive.druid.http.read.timeout</td>
<td>Read timeout period for the HTTP client in ISO8601 format. For example, P2W, P3M, PT1H30M, PT0.750S are possible values.</td>
</tr>
<tr>
<td>hive.druid.sleep.time</td>
<td>Sleep time between retries in ISO8601 format.</td>
</tr>
<tr>
<td>hive.druid.basePersistDirectory</td>
<td>Local temporary directory used to persist intermediate indexing state.</td>
</tr>
<tr>
<td>hive.druid.storage.storageDirectory</td>
<td>Deep storage location of Druid.</td>
</tr>
<tr>
<td>hive.druid.metadata.base</td>
<td>Default prefix for metadata table names.</td>
</tr>
<tr>
<td>hive.druid.metadata.db.type</td>
<td>Metadata database type. The only valid values are &quot;mysql&quot; and &quot;postgresql&quot;</td>
</tr>
<tr>
<td>hive.druid.metadata.uri</td>
<td>URI to connect to the database.</td>
</tr>
<tr>
<td>hive.druid.working.directory</td>
<td>Default HDFS working directory used to store some intermediate metadata.</td>
</tr>
<tr>
<td>hive.druid.maxTries</td>
<td>Maximum number of retries to connect to Druid before throwing an exception.</td>
</tr>
<tr>
<td>hive.druid.bitmap.type</td>
<td>Encoding algorithm used to encode the bitmaps.</td>
</tr>
</tbody>
</table>

If you installed both Hive and Druid with Ambari, then do not change any of the hive.druid.* properties other than those above when there are performance issues.

Related Information

Druid Integration