StorageHandlers

Hive Storage Handlers

Introduction

This page documents the storage handler support being added to Hive as part of work on HBaseIntegration. The motivation is to make it possible to allow Hive to access data stored and managed by other systems in a modular, extensible fashion.

Besides HBase, a storage handler implementation is also available for Hypertable, and others are being developed for Cassandra, Azure Table, JDBC (My SQL and others), MongoDB, ElasticSearch, Phoenix HBase, VoltDB and Google Spreadsheets. A Kafka handler demo is available.

Hive storage handler support builds on existing extensibility features in both Hadoop and Hive:

- input formats
- output formats
- serialization/deserialization libraries

Besides bundling these together, a storage handler can also implement a new metadata hook interface, allowing Hive DDL to be used for managing object definitions in both the Hive metastore and the other system's catalog simultaneously and consistently.

Terminology

Before storage handlers, Hive already had a concept of *managed* vs *external* tables. A managed table is one for which the definition is primarily managed in Hive's metastore, and for whose data storage Hive is responsible. An external table is one whose definition is managed in some external catalog, and whose data Hive does not own (i.e. it will not be deleted when the table is dropped).

Storage handlers introduce a distinction between *native* and *non-native* tables. A native table is one which Hive knows how to manage and access without a storage handler; a non-native table is one which requires a storage handler.

These two distinctions (*managed vs. external* and *native vs non-native*) are orthogonal. Hence, there are four possibilities for base tables:

- managed native: what you get by default with CREATE TABLE
- external native: what you get with CREATE EXTERNAL TABLE when no STORED BY clause is specified
- managed non-native: what you get with CREATE TABLE when a STORED BY clause is specified; Hive stores the definition in its metastore, but does not create any files itself; instead, it calls the storage handler with a request to create a corresponding object structure
- external non-native: what you get with CREATE EXTERNAL TABLE when a STORED BY clause is specified; Hive registers the definition in its metastore and calls the storage handler to check that it matches the primary definition in the other system

Note that we avoid the term *file-based* in these definitions, since the form of storage used by the other system is irrelevant.

DDL

Storage handlers are associated with a table when it is created via the new STORED BY clause, an alternative to the existing ROW FORMAT and STORED AS clause:

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [COMMENT col_comment], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [col_comment], col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name, ...)] INTO num_buckets BUCKETS]
  [
   [ROW FORMAT row_format] [STORED AS file_format]
   | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
  ]
  [LOCATION hdfs_path]
  [AS select_statement]
```

When STORED BY is specified, then row_format (DELIMITED or SERDE) and STORED AS cannot be specified. Optional SERDEPROPERTIES can be specified as part of the STORED BY clause and will be passed to the serde provided by the storage handler.

See CREATE TABLE and Row Format, Storage Format, and SerDe for more information.

Example:

```
CREATE TABLE hbase_table_1(key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES (
"hbase.columns.mapping" = "cf:string",
"hbase.table.name" = "hbase_table_0"
);
```

DROP TABLE works as usual, but ALTER TABLE is not yet supported for non-native tables.

Storage Handler Interface

The Java interface which must be implemented by a storage handler is reproduced below; for details, see the Javadoc in the code:

```
package org.apache.hadoop.hive.ql.metadata;

import java.util.Map;

import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.hive.metastore.HiveMetaHook;
import org.apache.hadoop.hive.ql.plan.TableDesc;
import org.apache.hadoop.hive.serde2.SerDe;
import org.apache.hadoop.mapred.InputFormat;
import org.apache.hadoop.mapred.OutputFormat;

public interface HiveStorageHandler extends Configurable {
  public Class<? extends InputFormat> getInputFormatClass();
  public Class<? extends OutputFormat> getOutputFormatClass();
  public Class<? extends SerDe> getSerDeClass();
  public HiveMetaHook getMetaHook();
  public void configureTableJobProperties(
    TableDesc tableDesc,
    Map<String, String> jobProperties);
}
```

The `HiveMetaHook` is optional, and described in the next section. If `getMetaHook` returns non-null, the returned object's methods will be invoked as part of metastore modification operations.

The `configureTableJobProperties` method is called as part of planning a job for execution by Hadoop. It is the responsibility of the storage handler to examine the table definition and set corresponding attributes on jobProperties. At execution time, only these jobProperties will be available to the input format, output format, and serde.

See also FilterPushdownDev to learn how a storage handler can participate in filter evaluation (to avoid full-table scans).

HiveMetaHook Interface

The `HiveMetaHook` interface is reproduced below; for details, see the Javadoc in the code:

```
package org.apache.hadoop.hive.metastore;

import org.apache.hadoop.hive.metastore.api.MetaException;
import org.apache.hadoop.hive.metastore.api.Partition;
import org.apache.hadoop.hive.metastore.api.Table;

public interface HiveMetaHook {
  public void preCreateTable(Table table)
    throws MetaException;
  public void rollbackCreateTable(Table table)
    throws MetaException;
  public void commitCreateTable(Table table)
    throws MetaException;
  public void preDropTable(Table table)
    throws MetaException;
  public void rollbackDropTable(Table table)
    throws MetaException;
  public void commitDropTable(Table table, boolean deleteData)
    throws MetaException;
```

Note that regardless of whether or not a remote Thrift metastore process is used in the Hive configuration, meta hook calls are always made from the Hive client JVM (never from the Thrift metastore server). This means that the jar containing the storage handler class needs to be available on the client, but not the thrift server.

Also note that there is no facility for two-phase commit in metadata transactions against the Hive metastore and the storage handler. As a result, there is a small window in which a crash during DDL can lead to the two systems getting out of sync.

Open Issues

- The storage handler class name is currently saved to the metastore via table property `storage_handler`; this should probably be a first-class attribute on MStorageDescriptor instead
- Names of helper classes such as input format and output format are saved into the metastore based on what the storage handler returns during CREATE TABLE; it would be better to leave these null in case they are changed later as part of a handler upgrade
- A dummy location is currently set for a non-native table (the same as if it were a native table), even though no Hive files are created for it. We would prefer to store null, but first we need to fix the way data source information is passed to input formats (HIVE-1133)
- Storage handlers are currently set only at the table level. We may want to allow them to be specified per partition, including support for a table spanning different storage handlers.
- FileSinkOperator should probably be refactored, since non-native tables aren't accessed as files, meaning a lot of the logic is irrelevant for them
- It may be a good idea to support temporary disablement of metastore hooks as part of manual catalog repair operations
- The CREATE TABLE grammar isn't quite as strict as the one given above; some changes are needed in order to prevent STORED BY and row_format both being specified at once
- CREATE TABLE AS SELECT is currently prohibited for creating a non-native table. It should be possible to support this, although it may not make sense for all storage handlers. For example, for HBase, it won't make sense until the storage handler is capable of automatically filling in column mappings.