Table Copy - background, incremental data load

Author: Rajat Venkatesh/Pavan Srinivas
Date: Oct 13 2014

Motivation

Traditionally, Hive and other tools in the Hadoop eco-system havent required a load stage. However, with recent developments, Hive is much more performant when data is stored in specific formats like ORC, Parquet, Avro etc. Technologies like Presto, also work much better with certain data formats. At the same time, data is generated or obtained from 3rd parties in non-optimal formats such as CSV, tab-limited or JSON. Many a times, its not an option to change the data format at the source. We've found that users either use sub-optimal formats or spend a large amount of effort creating and maintaining copies. We want to propose a new construct - Table Copy - to help “load” data into an optimal storage format. Table Copy will be typically linked with external tables.
In the rest of the document, Hive Table refers to the original table. It may or may not be partitioned.

Table Copies are different from load into traditional databases in the following ways:

- A load operation can happen in the background while queries are reading the Hive Table.
- A table copy will be used intelligently in SELECT queries. Data from a table copy will be used even though a query refers to the Hive Table.
- In a partitioned Hive table, a mixture of partitions from the Hive Table and Table Copy may be read. This may happen if a partition has not been loaded into the Table Copy.

Table Copies are not materialized views in the following ways:

- Table Copies will not store any derived columns.
- Table Copies are not maintained or synchronized like materialized views. If data is added or modified in a non-partitioned Hive Table or a partition, a “load” operation has to be triggered for the Table Copy to pick it up. Until then queries will see the loaded snapshot of the data.
- Table Copies will not store aggregate or join results.

Requirements

- GRANT/REVOKE privileges will be carried over from the base table. Permissions cannot be edited for a table copy.
- Partitioned tables are supported.
- Not all partitions have to be loaded into a *Table Copy*. If a partition is missing in a Table Copy, the partition from Hive Table will be read.

- There can be multiple copies of a table.
- If there are multiple Table Copies for a Hive Table, then they can be chosen by name or by the storage format.
- The feature can turned on or off.

User Surface

SQL Syntax

CREATE/DROP

```
CREATE TABLECOPY copy_name
ON TABLE base_table_name
[TCPROPERTIES (property_name=property_value, ...)]
[IN TABLE tcopy_table_name]
[STORED AS file_format ]
[SORTED BY (col_name, ...)]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
[COMMENT "some comment"]

file_format:
  : SEQUENCEFILE
  | TEXTFILE
  | RCFILE  (Note: Only available starting with Hive 0.6.0)
  | ORC     (Note: Only available starting with Hive 0.11.0)
  | INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

DROP TABLECOPY [IF EXISTS] copy_name ON base_table_name
```

Notes

- User can create many *table copies* for a *base table*.
- Inside the metastore, two objects are created.
    a. Managed Table that will store the data. The name can be specified by *index_table_name* or is generated as per a pattern.
    b. Index that ties the *base_table_name* to *index_table_name.* The name of the index is required and specified by *copy_name*.
- SERDE, Location, Sort Order and table properties can be specified.
- ROW FORMAT options for delimited/text data is NOT supported. I can add it but I dont see the point given that these are for fast access.
- Default file format is ORC

- By default the copy is NOT sorted. If SORTED BY is specified, then each partition is sorted.
- TCPROPERTIES will accept the following properties
    a. Retention Time: In terms of number of top-level partitions
    b. Max size: Number in GBs

LOAD TABLECOPY

```
LOAD TABLECOPY copy_name ON base_table_name [PARTITION partition_spec]

partition_spec:
  : (partition_col = partition_col_value,
     partition_col = partiton_col_value, ...
    )
```

Notes

- Generates a `INSERT OVERWRITE TABLE` for *every* partition. An example plan is shown in the appendix.
- If *partition_spec* is not specified then this will load all partitions which are missing in the table copy. If RETENTION or MAX is specified then these properties will be adhered to.

ALTER TABLECOPY

```
ALTER TABLECOPY copy_name ON base_table_name
 [ SET TCPROPERTIES (property_name=property_value, ...)
   | DROP [IF EXISTS] PARTITION partition_spec, PARTITION partition_spec,...
   | DROP EXPIRED PARTITIONS]
```

Notes

DROP EXPIRED PARTITIONS will drop unnecessary partitions by referring to TCPROPERTIES. The two criteria are number of partitions and size of the Table Copy.

SHOW TABLECOPY

```
SHOW TABLECOPY ON base_table_name
```

List all table copies on *base_table_name* with various properties listed.

DESCRIBE TABLECOPY

```
DESCRIBE TABLECOPY copy_name ON base_table_name
```

List all table copies on *base_table_name* with various properties listed.

Architecture and Design Details

Hive Internals

We will use the infrastructure laid out for INDEX tables. This is mainly the IDXs table in the metastore.

```
mysql> desc IDXS;
+---------------------+---------------+------+-----+---------+-------+
| Field               | Type          | Null | Key | Default | Extra |
+---------------------+---------------+------+-----+---------+-------+
| INDEX_ID            | bigint(20)    | NO   | PRI | NULL    |       |
| CREATE_TIME         | int(11)       | NO   |     | NULL    |       |
| DEFERRED_REBUILD    | bit(1)        | NO   |     | NULL    |       |
| INDEX_HANDLER_CLASS | varchar(4000) | YES  |     | NULL    |       |
| INDEX_NAME          | varchar(128)  | YES  | MUL | NULL    |       |
| INDEX_TBL_ID        | bigint(20)    | YES  | MUL | NULL    |       |
| LAST_ACCESS_TIME    | int(11)       | NO   |     | NULL    |       |
| ORIG_TBL_ID         | bigint(20)    | YES  | MUL | NULL    |       |
| SD_ID               | bigint(20)    | YES  | MUL | NULL    |       |
+---------------------+---------------+------+-----+---------+-------+
9 rows in set (0.01 sec)
```

Usage in SELECT queries

- In one of the transform phases, the optimizer will collect all the tables that are scanned and keep track of them in the Configuration object.
- For each table that has to be scanned
    a. The metastore client gets all the partitions that have to be scanned.
    b. All the table copies are obtained from the metastore.
    c. If a table copy has a partition with the same value as obtained in **a.**, then the table copy partition is returned instead.

There will be the following configuration parameters to control the above flow.

- **hive.metadata.tablecopy.enabled** - *true/false* - controls whether this feature is used or not.
- **hive.metadata.tablecopy.enabled.*table_name*** - *true/false* - controls whether this feature is enabled for a particular table.
- **hive.metadata.tablecopy.*table_name*** - *table copy name* - specifies the table copy name to be used for a particular table.
- **hive.metadata.tablecopy.type.*table_name*** - *InputFormat class name* - for a particular table, use a table copy which stores data using a specific format. For e.g. *org.apache.hadoop.hive.ql.io.orc.OrcInputFormat*

Appendix

Explain plan for loading ONE UNSORTED partition

```
hive> explain load tablecopy c1 on test_table_hive partition (dt='2012-03-12T00-00');
OK
ABSTRACT SYNTAX TREE:
  (TOK_LOADTBLCOPY test_table_hive c1 (TOK_PARTSPEC (TOK_PARTVAL dt '2012-03-12T00-00')))

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  null depends on stages: Stage-1
  Stage-2 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        test_table_hive
          TableScan
            alias: test_table_hive
            Select Operator
              expressions:
                    expr: a
                    type: int
                    expr: b
                    type: int
                    expr: c
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 1
          table:
                    input format: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
                    output format: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
                    serde: org.apache.hadoop.hive.ql.io.orc.OrcSerde
                    name: default.default__test_table_hive_c1__

  Stage: Stage-0
    Move Operator
      tables:
          partition:
            dt 2012-03-12T00-00
          replace: true
          table:
```

```
              input format: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
              output format: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
              serde: org.apache.hadoop.hive.ql.io.orc.OrcSerde
              name: default.default__test_table_hive_c1__

  Stage: null
```

Explain plan for loading ALL missing partitions

```
hive> explain load tablecopy c1 on test_table_hive;
OK
ABSTRACT SYNTAX TREE:
  (TOK_LOADTBLCOPY test_table_hive c1)

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  null depends on stages: Stage-1
  Stage-2 depends on stages: Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1,
Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1,
Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1, Stage-1
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  null depends on stages: Stage-1
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  null depends on stages: Stage-1
  Stage-1 is a root stage
```