Lab: HBase and Hive Integration

Objective: Understand how HBase and Hive integrate. You will complete data storage in HBase from Hive table data.

Data directory: `~/data/hbase/data`

What opportunities exist for deeper integration? Currently, customers are putting together solutions leveraging HBase, Phoenix, Hive etc. to build bespoke a closed-loop system for operational data and SQL analytics. We feel there is an opportunity to provide out-of-the-box integration with ease of use and additional capabilities such as transactions, cross datacenter failover etc.

So let’s take a look at Hive -> HBase integration.

What is ACID?

ACID stands for:

- Atomicity: a transaction should complete successfully or else it should fail completely i.e. it should not be left partially
- Consistency: any transaction will bring the database from one valid state to another state
- Isolation: every transaction should be independent of each other i.e. one transaction should not affect another
- Durability: if a transaction is completed, it should be preserved

in the database even if the machine state is lost or a system failure might occur

These ACID properties are essential for a transaction and every transaction should ensure that these properties are met.

Transactions in Hive

Transactions in Hive are introduced in Hive 0.13, but they only partially fulfill the ACID properties like atomicity, consistency, durability, at the partition level. Here, Isolation can be provided by turning on one of the locking mechanisms available with zookeeper or in memory.

But in Hive 0.14, new API’s have been added to completely fulfill the ACID properties while performing any transaction.

Transactions are provided at the row-level in Hive 0.14. The different row-level transactions available in Hive are as follows:

- Insert
- Delete
- Update

There are numerous limitations with the present transactions available in Hive.

ORC is the file format supported by Hive transaction. It is now essential to have ORC file format for performing transactions in Hive. The table needs to be bucketed in order to support transactions.

Missing Files

Make sure that the node you’ve picked is available to run `beeline`:

```
    beeline -n hive -u jdbc:hive2://localhost:10000
```

You may see something like this:

```
Connecting to jdbc:hive2://localhost:10000
18/09/25 03:56:01 [main]: WARN jdbc.HiveConnection: Failed
 to connect to localhost:10000
Could not open connection to the HS2 server. Please check
the server URI and if the URI is correct, then ask the adm
inistrator to check the server status.
Error: Could not open client transport with JDBC Uri: jdbc
:hive2://localhost:10000: java.net.ConnectException: Conne
ction refused (Connection refused) (state=08S01,code=0)
Beeline version 3.1.0.3.0.1.0-187 by Apache Hive
[centos@ip-172-30-12-85 ~]$
```

If so, try another node as a Hive client. Once you find a running node, you may need to copy the files from `data` on the Ambari (or other) node to your local drive. You can do an `scp` (a SSH copy of files) from the Ambari node like this:

```
[centos@ip-172-30-13-166 ~]$ sudo su -
[root@ip-172-30-13-166 ~]# scp -r /home/centos/data root@[
remote Hive node address]:/home/centos
```

> *Note: scp works the other way too*

Go to the Hive node, and you should now see your files:

```
[centos@ip-172-30-11-227 ~]$ ll
total 0
drwxrwxr-x. 5 root root 51 Sep 25 03:40 data
```

It may require you to `chown` the `data` directory as well:

```
[centos@ip-172-30-11-227 ~]$ sudo chown -R centos:centos d
ata/
```

Look at the Data

Do a `more` to look at the data in `iot_data.csv`:

```
    [centos@ip-10-0-0-237 data]$ more iot_data.csv
    _id,deviceParameter,deviceValue,deviceId,dateTime
    ObjectId(5a81b5395882b86112555f70),Temperature,27,SBS0
5,2018-02-12 15:39:37.050 UTC
    ObjectId(5a81b5395882b86112555f71),Humidity,59,SBS05,2
```

```
018-02-12 15:39:37.801 UTC
    ObjectId(5a81b53a5882b86112555f72),Sound,130,SBS04,201
8-02-12 15:39:38.629 UTC
    ObjectId(5a81b53b5882b86112555f73),Humidity,75,SBS05,2
018-02-12 15:39:39.272 UTC
    ObjectId(5a81b53b5882b86112555f74),Temperature,33,SBS0
2,2018-02-12 15:39:39.613 UTC
    ObjectId(5a81b53c5882b86112555f75),Sound,102,SBS03,201
8-02-12 15:39:40.363 UTC
    ObjectId(5a81b53c5882b86112555f76),Temperature,18,SBS0
2,2018-02-12 15:39:40.663 UTC
    ObjectId(5a81b53c5882b86112555f77),Flow,64,SBS05,2018-
02-12 15:39:40.678 UTC
    ObjectId(5a81b53d5882b86112555f78),Temperature,28,SBS0
4,2018-02-12 15:39:41.141 UTC
```

For bypassing any security issues, put the `iot_data.csv` table into the `tmp` directory in the Linux system:

```
    cp iot_data.csv /tmp/.
```

Let’s go ahead and create the anonymous user in HDFS:

```
sudo su hdfs
hdfs@host:~$ hdfs dfs -mkdir /user/anonymous
hdfs@host:~$ hdfs dfs -chown anonymous /user/anonymous
```

Create a Hive table and Load Data

We will begin by creating a Hive table with HBase charateristics. Notice the `hbase.table.name` below.

Now start beeline and create the table `iot_data`:

```
    CREATE EXTERNAL TABLE iot_data
        (rowkey string, parameter string, value int, device_id string, datetime string)
    ROW FORMAT SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe'
    STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    WITH SERDEPROPERTIES ("hbase.columns.mapping"=":key,para:param,
        para:value,para:device_id,para:datetime")
    TBLPROPERTIES ("hbase.table.name"="iot_data");
```

Easier to copy:

```
CREATE EXTERNAL TABLE iot_data (rowkey string, parameter string, value int, device_id string, datetime string) ROW FORMAT SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe' STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping"=":key,para:parameter,para:value,para:device_id,para:datetime")  TBLPROP
```

```
ERTIES ("hbase.table.name"="iot_data");
```

Validate the table in Hive:

```
    describe iot_data;
    INFO  : Compiling command(queryId=hive_20180815000016_
8eed1209-306c-4e5b-8e19-45fb250fae0a): describe iot_data
    INFO  : Semantic Analysis Completed (retrial = false)
    INFO  : Returning Hive schema: Schema(fieldSchemas:[Fi
eldSchema(name:col_name, type:string, comment:from deseria
lizer), FieldSchema(name:data_type, type:string, comment:f
rom deserializer), FieldSchema(name:comment, type:string,
comment:from deserializer)], properties:null)
    INFO  : Completed compiling command(queryId=hive_20180
815000016_8eed1209-306c-4e5b-8e19-45fb250fae0a); Time take
n: 0.03 seconds
    INFO  : Executing command(queryId=hive_20180815000016_
8eed1209-306c-4e5b-8e19-45fb250fae0a): describe iot_data
    INFO  : Starting task [Stage-0:DDL] in serial mode
    INFO  : Completed executing command(queryId=hive_20180
815000016_8eed1209-306c-4e5b-8e19-45fb250fae0a); Time take
n: 0.026 seconds
    INFO  : OK
    +------------+------------+----------+
    |  col_name  | data_type  | comment  |
    +------------+------------+----------+
    | id         | string     |          |
    | parameter  | string     |          |
    | value      | int        |          |
    | device_id  | string     |          |
    | datetime   | string     |          |
    +------------+------------+----------+
    5 rows selected (0.291 seconds)
```

Now we will create a temporary lookup table in Hive:

```
    CREATE TABLE iot_in (
        id string, parameter string, value string, device_id string, datetime string
        ) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
        WITH SERDEPROPERTIES ( "separatorChar" = ",", "quoteChar" = "\"")
        STORED AS TEXTFILE  location '/tmp/iot_data.csv';
```

Easier to copy:

```
CREATE TABLE iot_in (id string, parameter string, value string, device_id string, datetime string) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES ( "separatorChar" = ",", "quoteChar" = "\"") STORED AS TEXTFILE  location '/tmp/iot_data.csv';
```

And describe it:

```
    describe iot_in;
```

and should be empty:

```
    INFO  : OK
    +------------+------------------+---------------+----
---------------+------------------+
    | iot_in.id  | iot_in.parameter  | iot_in.value  | iot
_in.device_id  | iot_in.datetime  |
    +------------+------------------+---------------+----
---------------+------------------+
    +------------+------------------+---------------+----
---------------+------------------+
    No rows selected (0.121 seconds)
```

Now load data into the table:

```
    LOAD DATA LOCAL INPATH '/tmp/iot_data.csv' OVERWRITE I
NTO TABLE iot_in;
```

Now print to make sure its loaded:

```
select * from iot_in limit 20;
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldS
chema(name:iot_in.id, type:int, comment:null), FieldSchema
(name:iot_in.parameter, type:string, comment:null), FieldS
chema(name:iot_in.value, type:int, comment:null), FieldSch
ema(name:iot_in.device_id, type:string, comment:null), Fie
ldSchema(name:iot_in.datetime, type:string, comment:null)]
, properties:null)
INFO  : Completed compiling command(queryId=hive_201808150
33623_dcb145d1-b7e7-4af6-b4bd-4c78b4f6ee15); Time taken: 0
.209 seconds
INFO  : Executing command(queryId=hive_20180815033623_dcb1
45d1-b7e7-4af6-b4bd-4c78b4f6ee15): select * from iot_in li
mit 20
INFO  : Completed executing command(queryId=hive_201808150
33623_dcb145d1-b7e7-4af6-b4bd-4c78b4f6ee15); Time taken: 0
.001 seconds
INFO  : OK
+------------+-------------------+---------------+--------
-----------+------------------------------+
| iot_in.id  | iot_in.parameter  | iot_in.value  | iot_in.
device_id  |       iot_in.datetime        |
+------------+-------------------+---------------+--------
-----------+------------------------------+
| ObjectId(4 | deviceParameter   | NULL          | deviceI
d          | dateTime                     |
| ObjectId(3 | Temperature       | 27            | SBS05
           | 2018-02-12 15:39:37.050 UTC  |
| ObjectId(4 | Humidity          | 59            | SBS05
           | 2018-02-12 15:39:37.801 UTC  |
| ObjectId(7 | Sound             | 130            | SBS04
           | 2018-02-12 15:39:38.629 UTC  |
| ObjectId(6 | Humidity          | 75             | SBS05
           | 2018-02-12 15:39:39.272 UTC  |
| ObjectId(9 | Temperature       | 33             | SBS02
           | 2018-02-12 15:39:39.613 UTC  |
| ObjectId(8 | Sound             | 102            | SBS03
           | 2018-02-12 15:39:40.363 UTC  |
| ObjectId(2 | Temperature       | 18             | SBS02
           | 2018-02-12 15:39:40.663 UTC  |
| ObjectId(5 | Flow              | 64             | SBS05
           | 2018-02-12 15:39:40.678 UTC  |
| ObjectId(6 | Temperature       | 28             | SBS04
           | 2018-02-12 15:39:41.141 UTC  |
| ObjectId(1 | Humidity          | 69             | SBS03
           | 2018-02-12 15:39:41.804 UTC  |
| ObjectId(8 | Temperature       | 19             | SBS04
           | 2018-02-12 15:39:42.350 UTC  |
| ObjectId(6 | Temperature       | 28             | SBS05
           | 2018-02-12 15:39:42.593 UTC  |
| ObjectId(6 | Temperature       | 31             | SBS04
           | 2018-02-12 15:39:43.070 UTC  |
| ObjectId(2 | Sound             | 133            | SBS05
           | 2018-02-12 15:39:43.961 UTC  |
| ObjectId(6 | Flow              | 99             | SBS02
           | 2018-02-12 15:39:44.031 UTC  |
| ObjectId(4 | Humidity          | 65             | SBS04
           | 2018-02-12 15:39:44.667 UTC  |
| ObjectId(3 | Humidity          | 90             | SBS05
           | 2018-02-12 15:39:45.260 UTC  |
| ObjectId(4 | Flow              | 89             | SBS05
           | 2018-02-12 15:39:45.460 UTC  |
| ObjectId(4 | Sound             | 140            | SBS03
           | 2018-02-12 15:39:46.389 UTC  |
+------------+-------------------+---------------+--------
-----------+------------------------------+
20 rows selected (0.324 seconds)
```

Now, do a count:

```
0: jdbc:hive2://localhost:10000> select count(*) from iot_
in;
INFO  : Compiling command(queryId=hive_20180925055819_1077
c68f-8aa0-4f74-a7dd-e817f93bc3d7): select count(*) from io
t_in
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldS
chema(name:_c0, type:bigint, comment:null)], properties:nu
ll)
INFO  : Completed compiling command(queryId=hive_201809250
55819_1077c68f-8aa0-4f74-a7dd-e817f93bc3d7); Time taken: 0
.371 seconds
INFO  : Executing command(queryId=hive_20180925055819_1077
c68f-8aa0-4f74-a7dd-e817f93bc3d7): select count(*) from io
t_in
```

```
INFO  : Query ID = hive_20180925055819_1077c68f-8aa0-4f74-
a7dd-e817f93bc3d7
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Subscribed to counters: [] for queryId: hive_20180
925055819_1077c68f-8aa0-4f74-a7dd-e817f93bc3d7
INFO  : Session is already open
INFO  : Dag name: select count(*) from iot_in (Stage-1)
INFO  : Tez session was closed. Reopening...
INFO  : Session re-established.
INFO  : Session re-established.
INFO  : Status: Running (Executing on YARN cluster with Ap
p id application_1537813688041_0010)

----------------------------------------------------------
------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED
  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------
------------------------------------
Map 1 .......... container     SUCCEEDED      2          2
        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1
        0        0       0       0
----------------------------------------------------------
------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAP
```

```
SED TIME: 9.44 s
...
```

There is a ton of DAG and Map/Reduce action displayed here.

> *Question: how many records did the table have? 426,879?*

Exit Hive by executing a `!q`:

```
    !q
    Closing: 0: jdbc:hive2://master1.hdp.com:2181/default;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2
```

Validate the Table in HBase

Enter HBase shell:

```
    hbase shell
```

Use LIST command to check tables:

```
    hbase(main):005:0> list
    TABLE
    ...
    iot_data
    iot_in
    ...
```

Validate the table in HBase:

```
    hbase(main):003:0> describe 'iot_in'
    Table iot_in is ENABLED

    iot_in

    COLUMN FAMILIES DESCRIPTION

    {NAME => 'RAW', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE
 => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETE
    D_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DA
TA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIO
    NS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'R
OW', CACHE_INDEX_ON_WRITE => 'false', IN_MEMORY => 'false
    ', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_O
N_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => '
    true', BLOCKSIZE => '65536'}

    1 row(s)
    Took 0.1312 seconds
```

Now use SCAN to find if data exists:

```
    hbase(main):004:0> scan 'iot_in'
```

As it shows, there is no data in the table:

```
    ROW                          COLUMN+CELL

    0 row(s)
    Took 0.0559 seconds

    hbase(main):005:0>
```

Populate the Table in Hive with Tez

Go back to Hive, and run below command to set Hive engine as Tez:

```
    set hive.execution.engine=tez;
```

Execute below HiveQL to populate the table

```
    INSERT OVERWRITE TABLE iot_data
    SELECT iot_in.id, iot_in.device_id, iot_in.parameter,
iot_in.value, iot_in.datetime
    FROM iot_in WHERE iot_in.device_id='SBS05';
```

Easier to copy:

```
INSERT OVERWRITE TABLE iot_data SELECT iot_in.id, iot_in.device_id, iot_in.parameter, iot_in.value, iot_in.datetime FROM iot_in WHERE iot_in.device_id='SBS05';
```

and you should see something like this:

```
    Query ID = root_20140830123030_3fee9010-e712-4c44-89ec-1261c220e424
    Total jobs = 1
    Launching Job 1 out of 1
    Status: Running (application id: application_1409394057604_0003)
    Map 1: -/-
    Map 1: 0/1
    Map 1: 0/1
    Map 1: 0/1
    Map 1: 0/1
    Map 1: 0/1
    Map 1: 0/1
    Map 1: 1/1
    Status: Finished successfully
    OK
    Time taken: 24.005 seconds
```

Now check the data in Hive:

```
    select * from iot_data;
```

now the table should have about 85,394 rows.

Finally the HBase table (probably want to do a LIMIT):

```
hbase(main):004:0> scan 'iot_data' , {'LIMIT' => 10}
```

And get a final count on `iot_data` (in terminal):

```
[centos@ip-172-30-13-166 ~]$ hbase org.apache.hadoop.hbase
.mapreduce.RowCounter 'iot_data'
```

> *Question: where is the row counter?*

Improve Performance (optional)

Let’s play with the data a bit to see of we can improve performance. When populating a Hive/Hbase table the number of mappers is determined by the number of splits determined by the InputFormat used in the MapReduce job. In a typical InputFormat, it is directly proportional to the number of files and file sizes. Now suppose your HDFS block configuration is configured for 128MB(default size) and you have a files with 160MB size then it will occupy 2 block and then 2 mapper will get assigned based on the blocks. But suppose if you have 2 files with 30MB size (each file) then each file will occupy one block and mapper will get assignd based on that.

When you are working with a large number of small files, Hive uses CombineHiveInputFormat by default. In terms of MapReduce, it ultimately translates to using CombineFileInputFormat that creates virtual splits over multiple files, grouped by common node, rack when possible. The size of the combined split is determined by

```
mapred.max.split.size
or
mapreduce.input.fileinputformat.split.maxsize ( in yarn/MR
2);
```

So if you want to have less splits (less mapper action) you need to set this parameter higher.

So clear the table out:

```
    truncate table iot_in;
```

Now set the performance variable:

```
0: jdbc:hive2://localhost:10000> set mapreduce.input.filei
nputformat.split.maxsize;
+---------------------------------------------------------
+
|                           set
```

```
|
+---------------------------------------------------------
+
| mapreduce.input.fileinputformat.split.maxsize=256000000
|
+---------------------------------------------------------
+
1 row selected (0.008 seconds)
0: jdbc:hive2://localhost:10000>
```

> *Note the size of the max split*

And then do another `LOAD DATA LOCAL ...` command.

Now another `select count(*) ...` command

What happened? Did this run go faster?

Alternative (optional)

A newer approach to loading data in Hive/HBase is to use the ImportTsv tool:

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=, -Dimporttsv.columns="HBASE_ROW_KEY,para:deviceParameter,para:deviceValue,para:deviceId,para:dateTime" iot_data hdfs://[master node id]:/tmp/iot_data.csv
```

Summary

Now you have seen how to hook up Hive to HBase. The tables are interchangeable. Pretty cool, huh?