Microsoft

HBase Bulk Loader using Hive-HBase Integration Method on Azure

Prepared by

DM Jumpstart Engineering Team (askdmjfordmtools@microsoft.com)

Note: The detail provided in this document has been harvested as part of a customer engagement sponsored through the [Azure Data Services Jumpstart Program](#).

Table of Contents

Table of Figures

1   Introduction

Hive-HBase [integration](#) approach offers a way to use Hive SQL capabilities to administer and manage entities defined in HBase data store. This support is made possible by leveraging the [storage handler](#) framework that is integral to Hive. On the other hand, HBase offers a bulk-loader approach to ingest large volumes of data, likes of initial load without having to go through the compute-intensive standard RPC route.

At a high-level the bulk load process can be categorized into two broad phases or stages:

- Data preparation phase where source data format is migrated to HBase native file format – HFile.
- Data Ingress phase where in the generated HFiles are moved to the storage path managed by HBase (via HDFS-compliant storage layer).

The first step typically also includes transformation of data sets to accommodate downstream data requirements. For example, you may source discrete objects and join them in some order before generating the HFiles that represent your semantic grain for downstream. At a minimum, the process runs a simple map-reduce process, wherein based on your key design source data is read, organized by keys and by desired region splits, sorted and then persisted into immutable HFiles.

As a best practice the compute requirement for HFile generation must be sliced from clusters that may share same ADLS storage, but not the same cluster that also hosts your HBase. This document offers a step-by-step process explaining how one can migrate source data sets into HBase using Hive-HBase integration framework and the bulk-load approach.

2 General Guidelines

This document is aimed at a specific use case scenario associated data ingress into HBase. Hence, recommend reviewing following general guidelines about usage and how you can benefit from information shared in here:

- The document focuses on ingress optimization to HBase using the bulk-load approach, when HBase is configured using Azure HDInsight cluster setup.
- HDInsight offers different choices to determine the underlying at-rest storage layer. This document focuses on configuration that utilizes ADLS (Azure Data Lake Storage) as its primary storage layer.
- Hands-on exposure to HBase and general understanding about ADLS is essential to follow through the content. Recommend gaining some introduction into these toolkits before proceeding further.
- Besides, also recommend gaining insight to Hive-HBase integration framework that provides a SQL based approach to address steps to create the target HBase table.
  - You can create either a Hive managed HBase table or a Hive external table that points to an existing table in your target HBase cluster.
  - Ensure your Hive configurations point to your target HBase cluster.
- Considering the process described here is focused on Azure, it is important to also to gain familiarity with Azure networking concepts and other general cloud terminology. Example, the two clusters referred to in subsequent sections of this document are configured as part of same virtual network (VNET) and leverage a common ADLS resource.
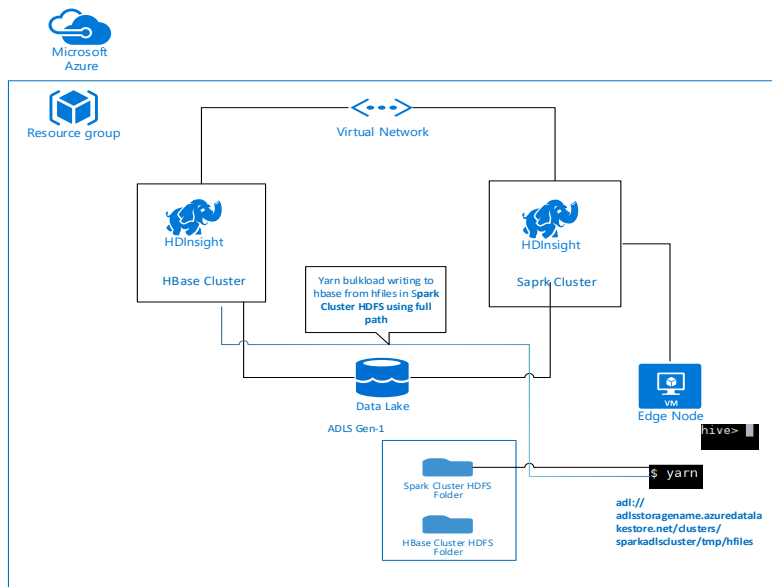
3   Empirical Overview

Figure 1: Bulk Load Process - Empirical View

Preceding visual offers an empirical view of the overall process that is followed, including the cluster setup. Scenario used in this demo is as follows:

- There are two HDInsight clusters used:
    a. HBase cluster, named as *hbaseadlscluster* and dedicated to just running HBase service and its dependent services such as Zookeeper, YARN, etc.
    b. Spark cluster with edge node attached to it, named as *sparkadlscluster*. Please note that use of Spark here is only a conventional reference, indicating some HDInsight cluster setup to address compute needs. Since generation of HFiles is also a compute intensive process, besides being storage heavy, it is recommended to address such compute on a separate cluster or set of fenced resources that do not conflict with HBase requirements for compute.
- Both *hbaseadlscluster* and *sparkadlscluster* are configured to use the same ADLS backend as their primary storage. Each have their own top-level folders – *hbaseadlscluster* and *sparkadlscluster* respectively. Other folders are used for storing source files data and other internal hive tables.
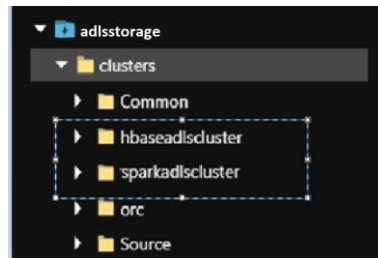
Figure 2: ADLS Storage Hierarchy sample

- All the steps mentioned in this document are executed from the edge node which is attached to the spark cluster.
- Creation of the HFiles are triggered from the Edge node and the yarn bulk load command is executed on the edge node which takes the HFiles generated in HDFS and loads them into the HBase table.

4   Prepare Configurations on Edge Node

This section focuses on sequence of steps that when performed in the order presented, should get your data ingested into HBase using the bulk-load feature.

4.1   Copy Hive Configurations to Edge Node

The first step in the process after the edge node is created, is moving the Hive related configuration files from HBase cluster head node to the edge node. We are taking all the files under /etc/hive/conf from the HBase cluster and copying them to spark edge node to the same folder(/etc/hive/conf). You can use tools like Linux `scp` command to copy files between the clusters. For a quick reference from your Linux bash shell, run `man scp` for quick reference on how to use the tool. Alternatively, you can depend on client-side tools such as FTP/SFTP tools. You have configured read-write permissions on source and target paths on respective cluster-nodes (VMs). You can ignore copying the `conf.server` folder and pick rest of the files.

```
sshuser@hn0-hbasea:~$ cd /etc/hive/conf
sshuser@hn0-hbasea:/etc/hive/conf$ ls -l
total 260
-rw-r--r-- 1 root root      1139 Jan 25 19:50 beeline-log4j.properties.template
drwx------ 2 hive hadoop    4096 Mar  4 08:43 conf.server
-rw-r--r-- 1 hive hadoop  198938 Jan 25 19:53 hive-default.xml.template
-rw-r--r-- 1 hive hadoop    2107 Mar  4 08:41 hive-env.sh
-rw-r--r-- 1 hive hadoop    2378 Jan 25 19:50 hive-env.sh.template
-rw-r--r-- 1 hive hadoop    3302 Mar  4 08:41 hive-exec-log4j.properties
-rw-r--r-- 1 hive hadoop    3474 Mar  4 08:41 hive-log4j.properties
-rw-r--r-- 1 hive hadoop   21840 Mar  4 08:41 hive-site.xml
-rw-r--r-- 1 root root      2049 Jan 25 19:50 ivysettings.xml
-rw-r--r-- 1 hive hadoop    7798 Mar  4 08:41 mapred-site.xml
-rw-r--r-- 1 hive hadoop    2662 Mar  4 08:41 parquet-logging.properties
sshuser@hn0-hbasea:/etc/hive/conf$ 
```

Figure 3: Hive configuration files

4.2   Copying HBase Configuration File to Edge Node

We need to copy the hbase-site.xml fie found in /etc/hbase/conf from the HBase cluster head node to the edge node attached to spark cluster. It must be copied to the same location(/etc/hbase/conf) in the edge node. Recommend using Linux SCP command to copy data between the nodes (VMs).

```
sshuser@hn0-hbasea:~$ cd /etc/hbase/conf
sshuser@hn0-hbasea:/etc/hbase/conf$ ls -l
total 40
-rw-r--r-- 1 hbase root   2966 Mar  4 08:41 hadoop-metrics2-hbase.properties
-rw-r--r-- 1 root  root   4537 Jan 25 18:57 hbase-env.cmd
-rw-r--r-- 1 hbase hadoop 3041 Mar  4 08:41 hbase-env.sh
-rw-r--r-- 1 hbase hadoop  367 Mar  4 08:41 hbase-policy.xml
-rw-r--r-- 1 hbase hadoop 7735 Mar  4 08:41 hbase-site.xml
-rw-r--r-- 1 hbase hadoop 4844 Mar  4 08:41 log4j.properties
-rw-r--r-- 1 hbase root    127 Mar  4 08:41 regionservers
sshuser@hn0-hbasea:/etc/hbase/conf$ 
```

Figure 4: HBase configuration files

4.3 Create HBase Tables (using Hive-shell on edge node)

Once the required configuration files are copied and setup on the edge node, it is time now to create the target HBase table and proceed towards accomplishing the first goal in the process, i.e., generate HFiles. Please review the following sub-sections here in the order they are presented. The steps must be performed from the edge node of the compute cluster itself.

4.3.1 Launch Hive Shell

From command line, launch hive shell using the command - `/usr/bin/hive`.

```
sshuser@ed11-kaiser:~$ /usr/bin/hive

Logging initialized using configuration in file:/etc/hive/2.6.5.3007-3/0/hive-log4j.properties
hive>
```

Figure 5: Launch Hive Shell

4.3.2 Apply Hive SET parameters

This approach considered using Map-Reduce as Hive's execution engine. You can also try with Tez (which is the default if not specified otherwise). Apply following SET commands as also shown in the visual below:

- SET hive.execution.engine=mr;
- SET hive.hbase.generatehfiles=true;
- SET hive.mapred.supports.subdirectories=TRUE;
- SET mapred.input.dir.recursive=TRUE;
- SET hive.hbase.generatehfiles=true;

```
sshuser@ed11-:~$ /usr/bin/hive

Logging initialized using configuration in file:/etc/hive/2.6.5.3007-3/0/hive-log4j.properties
hive> SET hive.execution.engine=mr;
hive> SET hive.hbase.generatehfiles=true;
hive> set hive.mapred.supports.subdirectories=TRUE;
hive> set mapred.input.dir.recursive=TRUE;
hive> SET hive.hbase.generatehfiles=true;
hive>
```

Figure 6: Apply Hive SET Parameters

4.3.3 Create Hive External Table (to read source data set)

An external table feature of Hive DDL allows you to read data in supported native formats, by overlaying the source with a chosen semantic structure. However, note that the ordering of columns must match the underlying data. For example, in a CSV formatted source data column types must match the types that you choose in your DDL. Here is the DDL to create the Hive external table:

```
CREATE EXTERNAL TABLE passwd_csv(userid STRING, uid INT, shell STRING)
ROW FORMAT
DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION 'adl://adlsstorage.azuredatalakestore.net/clusters/Source';
```

4.3.4 Create Hive External Table to preserve source data in ORC (Optional)

Converting source data to ORC will be helpful, especially if you consider re-purposing it for other down-stream needs besides ingesting data into HBase. For example, you can run data exploration checks using Hive SQL. And, Hive SQL works better on ORC, considering the benefits that ORC has to offer – columnar storage, compression and push-down predicates. You may choose to store data in other Hive-supported file formats like Avro, Parquet, etc. Otherwise, you can opt-out from considering this step in the sequence. Instead, you can source data from the external table created above directly into target HBase table, to be able to generate needed HFiles.

Note – using the keyword EXTERNAL you can manage the underlying storage of the ORC files in locations other than that managed by Hive (i.e., Hive data warehouse folders in a HDFS-compliant file system like ADLS).

Here is the DDL to create a Hive external table that maintains your source data in ORC format:

```
CREATE EXTERNAL TABLE passwd_orc(userid STRING, uid INT, shell STRING)
STORED AS ORC
LOCATION 'adl://adlsstorage.azuredatalakestore.net/clusters/orc';
```

Part of this preparation step is also to move your source data from its native CSV format into ORC. Here is the SQL Insert to accomplish just that:

```
INSERT INTO TABLE passwd_orc SELECT * FROM passwd_csv;
```

Here is the CLI snapshot that offers the result:

Figure 7: Source CSV to Hive ORC External table

4.3.5   Create HBase table using Hive Shell

To generate HFiles we need a reference to the target HBase table semantics that Hive-HBase integration can depend on. Here is the DDL to create a HBase table using Hive shell:

```
CREATE TABLE passwd_hbase(userid STRING, uid INT, shell STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,passwd:uid,passwd:shell');
```

Following visual provides a command line reference:

Figure 8: Create HBase table using Hive shell

4.3.6   Generate HFiles

Now that we have the semantic structures to both read the source data set and to write to the target HBase entity, let's now generate the HFiles using Hive shell. Since the outcome of this step is to be able to successfully generate HFiles and not yet updating them to the target HBase cluster, we need choose a location where Hive can persist the generated HFiles.

The location referenced here is on the primary storage configured for your Spark HDInsight cluster on which you launch the hive shell. Let's consider the location path as */tmp/hfiles_hbase/passwd*. It is not required to predefine the folder structure under /tmp. It is handled by Hive itself.

Figure 9: HDFS /tmp (to house generated HFiles)

The first step is to set the above chosen path to house the generated HFiles. Note in the above path, the leaf node *passwd* represents the column family under which your extracted column attributes will be packed as HFiles. This achieved by configuring the Hive SET command. Once the set command is applied, you can execute the INSERT to extract data from your source and generate HFiles. Here's the command sequence to run from the hive shell launched from your spark edge node:

```
SET hfile.family.path=/tmp/hfiles_hbase/passwd;
INSERT OVERWRITE TABLE passwd_hbase SELECT DISTINCT userid,uid,shell FROM passwd_orc CLUSTER BY userid;
```


Figure 10:Generating HFiles

The above sequence of commands produces the HFiles that contain your extracted source data, persisted to the HDFS path on your Spark HDFS cluster. Since the cluster is configured with ADLS as its primary storage, you could access the location from ADLS explorer as well. Now let's

verify the generated HFiles to exist in the path that is pointed in the *SET hfile.family.path* command above.

Figure 11: List of generated HFiles

4.3.7   Moving HFiles to target HBase Table

Once the HFiles are generated, the last mile step i.e., the second phase of the bulk-load process involves moving the files to storage layer managed by HBase. And, updating the HBase meta information working with the HBase Master. Information is further percolated up to each of the region servers (as part of HBase internal processes). The command here is the `yarn bulkload`. Following is the command snippet that allows you to accomplish this last step in the process:

```
$ export HADOOP_CLASSPATH=`hbase classpath`
$ yarn jar /usr/hdp/current/hbase-client/lib/hbase-server.jar completebulkload
adl://adlsstorage.azuredatalakestore.net/clusters/sparkadlscluster/tmp/hfiles_hbase passwd_hbase
```

The parameters to the command are described as follows:

- Completebulkload – this is an instruction to the hbase-server.jar to initialize the last step in the sequence to complete bulk load by moving the files pointed to by the path.
- HFile path – the ADL:// location is where the Hive-HBase integration generates the HFiles (the INSERT OVERWRITE... command). The path can be obtained from the Azure portal.
- Table name – the parameter passwd_hbase is the name of the HBase table to which the generated HFiles must be attached.

Once the above command concludes successfully, the HBase table can be queried for new records that you have sourced. The query can be run from within Hive shell that points to the HDInsight cluster hosting your HBase service, or from a HBase shell.

Figure 12: Validate data in target HBase table

5   Appendices

5.4   Appendix - Known Limitations

- Hive-HBase integration
  - o [Open JIRA issues](#); check specifically on ability to load data by creating region pre-splits ([HIVE-13584](#)). Current work around is to try creating the table first in HBase, then define an external table using Hive-HBase integration approach.

5.5   Appendix - External References

- [HBase Reference Guide](#)
- [Azure HDInsight Pricing](#)
- [Azure Data Lake Storage Gen1 Pricing](#)
- [Quick Reference to HBase Shell Commands](#)
- [HBase and HDFS: Understanding file system usage](#)
- [HBase Default configuration XML](#)
- [HDFS Default configuration XML](#)
- [Apache HBase in HDInsight: Overview](#)
- [Apache Hadoop Component versions in HDInsight](#)
- HFile Format [Git Resource](#) & [Book Reference](#)
- [HBase Architecture 101](#) (Old article but useful reference)
- [HBase zookeeper znodes explained](#)
- [HDInsight HBase: 9 things you must do to get great HBase Performance](#)
- [HDInsight – How to perform Bulk Load with Phoenix?](#)
- [About HBase flushes and compactions](#)
- [Apache HBase and Apache Phoenix, more on block encoding and compression](#)
- [Offheap Read-Path in Production – The Alibaba Story](#)
- [How are bloom-filters used in HBase](#)
- [How to list regions and their sizes for a HBase table](#)
- [HBase and HDFS: Understanding Filesystem Usage in HBase](#)
- [Hive-HBase integration](#)

6   Feedback and suggestions

If you have feedback or suggestions for improving this data migration asset, please contact the Data Migration Jumpstart Team (askdmjfordmtools@microsoft.com). Thanks for your support!

Note: For additional information about migrating various source databases to Azure, see the Azure Database Migration Guide.