Denodo HBase Custom Wrapper

Revision 20140904

CONTENTS

1 INTRODUCTION

The `hbase-customwrapper` is a Virtual DataPort custom wrapper that enables VDP to perform read operations on an HBase database.

HBase is a column-oriented **NoSQL** database management system that runs on top of Hadoop infrastructure, that allows you to store and process large amounts of data on multiple machines.

As usually happens in the NoSQL database world, HBase does not support SQL queries, therefore it is necessary to use a binary client API to access to the database system.

NOTE: *The scope of this wrapper is limited because it depends on the specific version and configuration of the HBase server you are accessing. Besides, there are direct dependencies between the binary files of the server and the client. For this reason, it is likely that it will be necessary to perform minor code-level modifications to the wrapper in order to make it compatible with specific HBase installations: version of the API client libraries, configuration entries at the `hbase-site.xml` file, etc.*
Also note that this wrapper contains the HBase client libraries themselves (a requirement of HBase), so in order to guarantee the wrapper will work, the machine on which VDP runs must meet all the HBase client libraries configuration requirements (network reachability, DNS configuration, etc.). Said in other words: this wrapper should work OK if a standalone java application using the HBase client libraries works OK, in the same machine.

2 FEATURES

2.1 JAVA ARTIFACT ORGANIZATION

This wrapper uses the HBase Java client API, which in turn has a dependency on the `hadoop common`, `zookeeper`, and `thrift` libraries. These binary dependencies have two undesirable, but unavoidable effects:

1. Increase the size of the binary artifacts deployed into Virtual DataPort.
2. Un-generalize the custom wrapper, by binding it (and its distribution) to specific Hadoop/HBase software versions.

2.2 ARCHITECTURE

The HBase architecture has two main components:

1. One **HMaster** node/process that is responsible for coordinating the cluster and execute administrative operations like region allocation and failover, log splitting, and load balancing.

1. One or several **HRegionServers** that are responsible for handling a subset of the table's data and I/O requests to the hosting regions, flushing the in-memory data store (MemStore) to HDFS, and splitting and compacting regions.

The HBase client provides the wrapper with the required API to access to the HBase cluster. The ZooKeeper cluster acts as a coordination service for the entire HBase cluster, handling master selection, root region server lookup, node registration, configuration information, hierarchical naming space, and so on.

The next diagram provides a general overview of the system architecture:
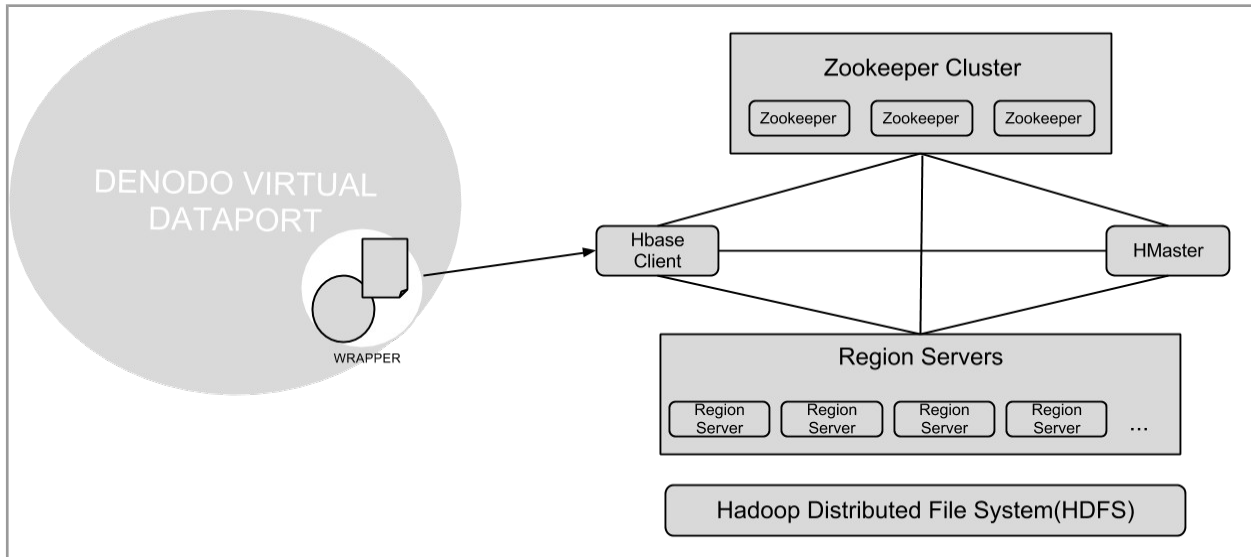
2.3 DATA MODEL

1. **Tables** in HBase are made up of rows and columns.
2. **Columns** are grouped into family columns. A column name is made of its column family name and a qualifier `familyName:columnName`.
3. **Column families** regroup data of the same nature; they are stored together on the filesystem.
4. **Rows** have one row key. They are lexicographically sorted with the lowest order appearing first in a table.
5. **Cells** are identified by row, column and version. The cell contents are an uninterpreted array of bytes.
6. **Versions** are cells that have the same column and table, every version has associated a timestamp, that it is a long (milliseconds). The HBase version dimension is stored in decreasing order, the latest value is found first. The number of versions that we can fetch when a cell is retrieved is configurable. HBase, by default, returns the latest version.

You can read more about the data model in the next link: http://hbase.apache.org/book/datamodel.html.

2.4 CAPABILITIES

This wrapper allows only read operations. It can delegate to HBase the following query artifacts and operators:

1. Operators: =, <>, REGEXP_LIKE, LIKE, IN, IS TRUE, IS FALSE, IS NULL, IS NOT NULL, CONTAINS_AND, CONTAINS_OR.
2. AND operations.
3. OR operations.
4. StartRow and StopRow: HBase stores rows sorted lexicographically by comparing the key bytes arrays. We can perform a scan to effectively select the start row and the stop row. So, HBase can return all the rows that pass the filter without having to read the whole table. Star row is **inclusive** while stop row is **exclusive**.
5. Data types: Text, Long, Integer, Float, Double, Boolean.
6. Projection of row_key column and column families but not individual columns.

Due to the delegation of the previous operators, the performance of the custom wrapper is better because many operations will be performed by HBase itself, instead of having to rely on VDP in-memory post-filtering.

2.5 <u>LIMITATIONS</u>

There are some limitations which should be taken into account when using this wrapper:

1. Read-only. Update and delete operations could be developed in the future, but they are not available as of current versions.
2. Row keys are uninterpreted bytes, so we have to be careful if we want to make VDP read data types different to the string one.
3. The <, <=, >, => operators cannot be delegated to HBase because its API-driven comparison operations are byte-only based, which makes it unsuitable for comparing numbers.
4. The wrapper cannot make the most out of the possibilities of HBase because of the difference in paradigm between this column-oriented NoSQL system and the relational paradigm VDP implements.

3 USAGE

In order to use the `hbase-customwrapper`, first it is necessary to add its jar file (the `jar-with-dependencies` version) into the VDPAdmin Tool in `File → Extensions`.

After this, a data source should be created in `File → DataSource → Custom`, selecting the pertinent jar, and writing a name for the data source. The custom wrapper is implemented by class com.denodo.connect.hadoop.hbase.HBaseConnector, and it will be shown when selecting the jar in the field `Class name` in this view.

The `hbase-customwrapper` includes the configuration file `hbase-site.xml`, with the default values commented out. When required we can change any of these parameter uncommenting that specific attribute and changing its value. For instance, when using the **Hortonworks Sandbox** it is necessary to **change the property** zookeeper.znode.parent from its value by default /hbase to /hbase-unsecure.

3.1 CREATING BASE VIEWS

The base views created from the `HBaseConnector` need the following **mandatory** parameters:

- `hbaseIP`: Name of the HBase machine or its IP address. It can be a list of HBase IPs separated by commas.
- `tableName`: HBase table from which we want to extract the data.
- `tableMapping`: A fragment of JSON, giving information about the queried HBase data structure and defining the elements, the column families and columns, that will be projected into the base view.

  Curly braces (`{`) must be escaped with a backslash (`\`).

For example:

```
\{
     'post': \{
          'title': 'text',
          'body': 'text'
     \},
     'image': \{
          'bodyimage': 'text',
          'header': 'text',
          'active': 'boolean'
     \}
\}
```

where `post` and `image` are column families and the other sub-elements are columns.

There is also an special case that allows retrieving only the `row_key` field of a data set, for performance reasons:

```
\{
   'row_key': 'text'
\}
```

And two **optional** parameters:

● `hbasePort`: ZooKeeper port, default is 2181.
● `cachingSize`: the number of rows that a scanner will fetch at once. If `cachingSize` is not set, we will use the caching value of the HTable we want to access.
   Higher caching values will enable faster scanners but will use more memory.

3.2 EXAMPLE

In the next example we want to extract the rows of the table `blogposts` table which `title` value (in the `post` family), starts with `'Hello World'`. So, in the HBase Shell we execute the next command.

```
hbase(main):001:0> scan 'blogposts', {FILTER =>
 "(SingleColumnValueFilter('post','title',=,'regexstring:Hello World'))"}
```

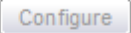In the following image we can see the results of this query:

```
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.2.21, rUnknown, Thu Jan 10 03:45:56 PST 2013

hbase(main):001:0> scan 'blogposts', {FILTER => "(SingleColumnValueFilter('post','title',=,'regexstring:Hello World'))"}
ROW COLUMN+CELL
post1 column=image:active, timestamp=1368536115329, value=true
post1 column=image:bodyimage, timestamp=1368536102858, value=5
post1 column=image:header, timestamp=1368536090247, value=image1
.jpg
post1 column=post:author, timestamp=1368536076921, value=The Aut
hor
post1 column=post:body, timestamp=1368536083159, value=This is a
blog post
post1 column=post:title, timestamp=1368536070847, value=Hello Wo
rld
post2 column=image:active, timestamp=1368537884925, value=false
post2 column=image:bodyimage, timestamp=1368537896082, value=0
post2 column=image:header, timestamp=1368537831101, value=imghea
der2
post2 column=post:author, timestamp=1368537743990, value=PLC
post2 column=post:body, timestamp=1368537800137, value=This is t
he second history
post2 column=post:title, timestamp=1368537706887, value=Hello Wo
rld2
post3 column=image:bodyimage, timestamp=1368537903556, value=10
post3 column=image:header, timestamp=1368537838765, value=imghea
der3
post3 column=post:title, timestamp=1368537714692, value=Hello Wo
rld3
3 row(s) in 2.7590 seconds

hbase(main):002:0>
```

It is possible to obtain the same result using VDP and the `hbase-customwrapper`.

Using the JSON fragment we saw in a previous section as table mapping, we create a base view:

hbaseIP	melkus.denodo.com
hbasePort	2181
tableName	blogposts
tableMapping	'post': \{ 'title': 'text', 'body': 'text' \}, 'image': \{ 'bodyimage': 'text', 'header': 'text', 'active': 'boolean' \}
cachingSize	
Kerberos enabled	☐
Kerberos principal name	
Kerberos keytab file	None ▼ [Configure]
Kerberos password	
Kerberos Distribution Center	

hbase_test	
row_key	text
⊟ post	⛁ hbase_test_post
body	text
title	text
⊟ image	⛁ hbase_test_image
bodyimage	text
active	boolean
header	text
start_row	text
stop_row	text

Once the base view is created, it can be used in VDP just like any other views. Let's see a VQL sentence equivalent to the above HBase shell command:

```
SELECT * FROM hbase_test WHERE (post).title like 'Hello World%'
```

row_key	post	image
post1	[Register]...	[Register]...
post2	[Register]...	[Register]...
post3	[Register]...	[Register]...

Total rows received: 3 (shown 3)

From the row_key values, we can check that the same results are obtained as if the HBase Shell had been used instead.

The previous image does not allow to see the array itself, it would require flattening, so here it is for the first returned tuple:

RESULT -> post

body	title
This is a blog post	Hello World

4 SECURE CLUSTER WITH KERBEROS

The configuration required for accessing a Hadoop cluster with Kerberos enabled is the same as the one needed to access HBase and, additionally, the user must supplied the Kerberos credentials.

The Kerberos parameters are:

- `Kerberos enabled`: Check it when accessing a Hadoop cluster with Kerberos enabled.
- `Kerberos principal name`: Kerberos v5 Principal name to access HBase, e.g. primary/instance@realm
- `Kerberos keytab file`: Keytab file containing the key of the Kerberos principal.
- `Kerberos password`: Password associated with the principal.
- `Kerberos Distribution Center`: Kerberos Key Distribution Center.

`hbase-customwrapper` provides two ways for accessing a kerberized Hadoop cluster:

1. The client has a valid Kerberos ticket in the ticket cache (obtained, for example, using the `kinit` command in the Kerberos Client).
   In this case only the `Kerberos enabled` parameter should be checked. The HBase wrapper would use the Kerberos ticket to authenticate itself against the Hadoop cluster.

2. The client does not have a valid Kerberos ticket.
   In this scenario all the Kerberos configuration parameters should be provided. `Kerberos keytab file` and `Kerberos password` are **mutually exclusive**.

hbaseIP	melkus.denodo.com
hbasePort	2181
tableName	analytics_demo
tableMapping	'total': 'text',
cachingSize	
Kerberos enabled	✔
Kerberos principal name	test\@DENODO.COM
Kerberos keytab file	None ▼ [Configure]
Kerberos password	● ● ● ● ● ● ●
Kerberos Distribution Center	melkus.denodo.com

View edition

5 SOFTWARE REQUIREMENTS

`hbase-customwrapper` has been tested in Cloudera QuickStart VM 5.0.0 using Hadoop v2.4.0, HBase v0.96.2-hadoop2 and ZooKeeper v3.4.5.

Cloudera provides a virtual machine that gives a working Apache Hadoop environment out-of-the-box. It can be downloaded from:
http://www.cloudera.com/content/support/en/downloads.html

6 TROUBLESHOOTING

Symptom

Error message: "The node /hbase (or /hbase-unsecure) is not in ZooKeeper. It should have been written by the master. Check the value configured in 'zookeeper.znode.parent'. There could be a mismatch with the one configured in the master."

Resolution

Edit the configuration file `hbase-site.xml` included in the `hbase-customwrapper` jar and change the property zookeeper.znode.parent from its value /hbase to /hbase-unsecure (or viceversa).

Symptom

Error message: "com.google.protobuf.ServiceException: java.io.IOException: Call to `<your domain/your IP:your port>` failed on local exception: java.io.EOFException".

To see the real error message enable Hadoop **debugging** by adding these settings to VDP `log4j.xml`:

```
<logger name="org.apache.hadoop">
  <level value="debug"/>
</logger>
```

And you will see the following message in VDP log file: "org.apache.hadoop.ipc.RpcClient - IPC Client (...) exception { exception_class_name: "org.apache.hadoop.security.AccessControlException" stack_trace: "Authentication is required" do_not_retry: false } (...)"

Resolution

You are trying to connect to a Kerberos-enabled Hadoop cluster. You should configure the custom wrapper accordingly. See Secure cluster with Kerberos for **configuring Kerberos** on this custom wrapper.

Symptom

Error message: "com.google.protobuf.ServiceException: java.io.IOException: Couldn't setup connection for <Kerberos principal> to <HBase principal>".

To see the real error message enable Hadoop **debugging** by adding these settings to VDP `log4j.xml`:

```
<logger name="org.apache.hadoop">
  <level value="debug"/>
</logger>
```

And you will see the following message in VDP log file: org.apache.hadoop.ipc.RpcClient - (...) Server not found in Kerberos database (7) - UNKNOWN_SERVER)]

Resolution

Check that `nslookup` is returning the fully qualified hostname of the KDC. If not, modify the `/etc/hosts` of the client machine for the KDC entry to be of the form `"IP address fully.qualified.hostname alias"`.

Symptom

Error message: "org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient permissions for user '<user>' for scanner open on table <table>"

Resolution

After securing an HBase cluster, if user has not be given appropriate privileges, HBase access will fail with an error "Insufficient permissions for user". It is an expected behavior.

Once the cluster has been secured, a user has to authenticate itself to Kerberos by doing a `kinit`. By default, hbase is a superuser who was full access and can be used to grant privileges to other users. So you can use hbase keytab file to perform a `kinit`.

```
$ sudo -u hbase kinit -kt
/var/run/cloudera-scm-agent/process/305-hbase-MASTER/hbase.keytab
hbase/melkus.denodo.com@DENODO.COM
```

Now, after successful `kinit`, you can login to HBase shell and grant privileges to other users.

```
$ sudo -u hbase hbase shell
14/07/17 04:27:19 INFO Configuration.deprecation: hadoop.native.lib
is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.96.1.1-cdh5.0.0, rUnknown, Thu Mar 27 23:03:17 PDT 2014

hbase(main):001:0> grant 'test','R'
0 row(s) in 5.4110 seconds
```