

The Vertica Connector for Hadoop

Hadoop uses an API called InputFormat to read information from a data source such as HDFS or a database and OutputFormat to write information to a data target. The Vertica Hadoop connector is an implementation of the Input and Output Format APIs optimized for the Vertica Analytic Database. This document describes how to use the Vertica Input and Output Formats for java based Hadoop jobs, with Hadoop streaming and with Pig, a SQL-like language that runs on top of Hadoop.

Obtaining and Deploying the Vertica Hadoop Connector

The Vertica Hadoop connector is available from www.vertica.com/hadoop. The zip file contains three jar files:

1. Hadoop Connector: hadoop-vertica.jar
2. Hadoop Connector example: hadoop-vertica-example.jar
3. Pig Connector: pig-vertica.jar

All the jar files contain source code as well.

You only need to place the vertica jdbc driver jar in the \$HADOOP_HOME/lib directory on each node in the Hadoop cluster. To install the Vertica Hadoop connector, place just the hadoop-vertica.jar in the \$HADOOP_HOME/lib directory on each node. To install the Vertica Pig connector, place pig-vertica.jar in the same directory as environment variable PIG_CLASSPATH. Alternatively, add the directory where pig-vertica.jar is saved to PIG_CLASSPATH environment variable.

Connecting to Vertica from Hadoop Java Jobs

The Vertica connector implements an InputFormat and OutputFormat as well as a corresponding VerticaRecord for reading and writing data from Hadoop. To configure a job using the Vertica Hadoop connector, use the following settings on a JobConf object:

```
jobConf.setInputFormat(VerticaInputFormat.class);
VerticaConfiguration.configureVertica(conf, new String[]{"host1", "host2", "host3"}, "dbname",
"dbport", "dbuser", "dbpass");
```

The first function call sets the input format to use the VerticaInputFormat class. The second function call configures the Vertica Hadoop connector to connect to any of the listed database hosts using the specified database, username and password. If there is no password, provide an empty string. The Vertica Hadoop connector will randomly select a host for each map or reduce task. To use a different database for output, use the following API call:

```
VerticaConfiguration.configureVertica(conf, new String[]{"host1", "host2", "host3"}, "dbname",
"dbport", "dbuser", "dbpass", new String[]{"output_host1", "output_host2", "output_host3"},
"output_dbname", "output_dbport", "output_dbuser", "output_dbpass");
```

Reading Data from Vertica

To specify a basic input query use the following API call:

```
//a simple query with an order by that will be split using limit and offset
VerticaInputFormat.setInput(jobConf, "select * from table order by key");
```

The basic input uses a single query with an order by. The Vertica Hadoop connector will create a number of InputSplit equal to the mappers specified in the `mapred.map.tasks` Hadoop configuration. Each runs a query using LIMIT and OFFSET to retrieve a different part of the table. This is the least efficient mechanism for querying Vertica.

A powerful interface that the Vertica Hadoop connector provides is to parameterize the input query and include a list of parameters as follows:

```
//a parameterized query with a literal list of parameters
VerticalInputFormat.setInput(conf, "select * from table where key = ?", "0", "1", "2");

//a parameterized query with a collection of parameters
Collection<List<Object>> params = new HashSet<List<Object>>{};
List<Object> param = new ArrayList<Object>();

//each param can have multiple arguments
param.add(new Integer(0));

params.add(param);
VerticalInputFormat.setInput(conf, "select * from table where key = ?", params);
```

In this case, the Vertica Hadoop connector will create an InputSplit for each parameter and each map task will run a single query substituting the specified parameter using JDBC prepared statements. The parameter list can include integers, dates or strings. When using literal strings the parameter should include single quotes such as `„string parameter 1“`, `„string parameter 2“`

Finally, the Vertica Hadoop connector accepts a parameter query that returns a result set to use as parameters for the query. The Vertica Hadoop connector runs the parameters query and then creates an InputSplit for each result record. From there the Vertica Hadoop connector sends queries similarly to when specifying the parameters directly.

```
//a parameterized query with a query source as a parameter
VerticalInputFormat.setInput(conf, "select * from table where key = ?",
    "select distinct key from table");
```

The Vertica Hadoop connector provides informational logging and error checking. In the case of errors such as invalid database configuration or errors in the query, the connector will throw an appropriate exception.

The mapper class will be defined as follows with any key value pair K, V as output. If there is no reducer class, the output key, value must be Text, VerticaRecord.

```
public static class Map extends
    Mapper<LongWritable, VerticaRecord, Text, DoubleWritable> {

    public void map(LongWritable key, VerticaRecord value, Context context)
        throws IOException, InterruptedException {
    }
}
}
```

Outputting to Vertica from Hadoop

The Vertica Hadoop connector allows for outputting from a Hadoop job to a table in Vertica. The table does not need to exist in advance and can be optimized directly from the connector. To output to a Vertica table specifying the following configurations:

```
jobConf.setOutputKeyClass(Text.class);
jobConf.setOutputValueClass(VerticaRecord.class);
VerticaOutputFormat.setOutput(jobConf, "table", <truncate>, "a int", "b boolean", "c char(1)", "d
date", "f float", "n numeric", "t timestamp", "v varchar(25)", "z varbinary");
```

At configuration time the Vertica Hadoop connector validates that the table exists and creates it if it does not. If the <truncate> argument is true, then the table is emptied before loading. At runtime each reducer (or mapper if numReducers = 0) opens a connection to a randomly selected host on the target database and streams in the results of the job.

The reducer class must be defined as follows with any key value pair K, V as input:

```
public static class Reduce extends
    Reducer<Text, DoubleWritable, Text, VerticaRecord> {
    VerticaRecord record = null;

    public void setup(Context context) throws IOException,
    InterruptedException {
        super.setup(context);
        try {
            record = VerticaOutputFormat.getValue(context.getConfiguration());
        } catch (Exception e) {
            throw new IOException(e);
        }
    }

    public void reduce(Text key, Iterable<DoubleWritable> values,
    Context context)
        throws IOException, InterruptedException {
    }
}
```

The output key must always be the name of the table specified in the `VerticaOutputFormat.setOutput` method call.

The VerticaRecord class

Each input and output record is defined using the `VerticaRecord` class. This class provides JDBC like access to data as well as metadata information such as column name and data type. All parameters to the class are zero indexed. For example, the first value is retrieved using `record.get(0)` and set using `record.set(0, value)`. The `set` method takes an optional third argument which will validate the data on set before attempting to coerce it for load.

Hadoop Streaming for Vertica

The Vertica Hadoop connector also implements Input and Output formatters that use the streaming interface. The streaming protocol uses as tab delimited key, value pair. In order to handle data that includes tabs and newlines, the Vertica Hadoop connector uses 0xa and 0xb as the default delimiter and terminator for input and output records. To run a Hadoop streaming job using the Vertica connector run the following command:

```
$HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-0.20.1-streaming.jar
  -Dmapred.vertica.output.table.name=table
  -Dmapred.vertica.hostnames=host1,host2,host3
  -Dmapred.vertica.database=db
  -Dmapred.vertica.username=user
  -Dmapred.vertica.input.query="select * from table order by key"
  -Dmapred.vertica.output.delimiter=,
  -inputformat com.vertica.util.hadoop.VerticaStreamingInput
  -outputformat com.vertica.util.hadoop.VerticaStreamingOutput
  -input /tmp/input
  -output /tmp/output
  -reducer /bin/cat
  -mapper /bin/cat
```

The `-inputformat` and `-Dmapred.vertica.input` arguments are only required when reading data from Vertica and the `-outputformat` and `-Dmapred.vertica.output.*` arguments are only required when writing data to Vertica. By default records sent to the mapper and expected by the reducer use the 0x7 character as the delimiter for columns in the value. Strings are not enclosed by quotation marks.

Connecting to Vertica from Pig

Vertica also provides a connector for loading and storing data from Pig. Deploy the Vertica Pig connector by placing the `hadoop-vertica.jar` and the `pig-vertica.jar` in the `$HADOOP_HOME/lib` directory on all nodes. Load data into Pig from Vertica using one of the following command:

```
--load a simple query
A = LOAD „sql://{select * from table order by key}“ USING
org.apache.hadoop.vertica.VerticaLoader(„host1,host2,host2“, „db“, „dbuser“, „dbpass“);
--load a query with constant parameters
A = LOAD „sql://{select * from table where key = ?};{1,2,3}“ USING
org.apache.hadoop.vertica.VerticaLoader(„host1,host2,host2“, „db“, „dbuser“, „dbpass“);
--load a query using a parameter query
A = LOAD „sql://{select * from table where key = ?};sql://{select distinct key from table}“ USING
org.apache.hadoop.vertica.VerticaLoader(„host1,host2,host2“, „db“, „dbuser“, „dbpass“);
```

The `dbpass` argument is optional. When specifying literal strings, be sure to escape the single quotes such as `\“string\“`.

To store data into Pig use the following load statement:

```
STORE A INTO „{table(a int, b boolean, c char(1), d date, f float, n numeric, t timestamp, v varchar(10), z varbinary)}“ USING org.apache.hadoop.vertica.VerticaStorer(„host1,host2,host2“, „db“, „dbuser“, „dbpass“);
```

If the table exists the STORE statement can include just the table name without the definition. As with input the dbpass argument is optional.

Vertica Hadoop Connector Parameters

The Vertica Hadoop connector uses the following parameters for configuration:

Parameter Name	Description	Required	Default
mapred.vertica.hostnames	Host names to connect to, selected from at random	Yes	
mapred.vertica.port	Port to connect to	No	5433
mapred.vertica.database	Name of database to connect to	Yes	
mapred.vertica.username	Username for Vertica	Yes	
mapred.vertica.password	Password for Vertica	No	empty
mapred.vertica.hostnames.output	Optional host names to connect to for output, selected from at random	No	Input hostnames
mapred.vertica.port.output	Optional port to connect to for output	No	5433
mapred.vertica.database.output	Optional name of database to connect to for output	No	Input database
mapred.vertica.username.output	Optional username for Vertica output	No	Input username
mapred.vertica.password.output	Optional password for Vertica output	No	Input password
mapred.vertica.input.query	Query to run for input	For input	
mapred.vertica.input.query.paramquery	Query to run to retrieve parameters	paramquery or params for input	

Parameter Name	Description	Required	Default
mapred.vertica.input.query.params	Static parameters for query	paramquery or params for input	
mapred.vertica.input.delimiter	Optional input delimiter for streaming	No	
mapred.vertica.input.terminator	Optional input terminator for streaming	No	
mapred.vertica.date_as_string	Whether to marshal dates as strings	No	
mapred.vertica.output.table.name	Output table name	For output	
mapred.vertica.output.table.def	Definition of output table types	For output to new tables	Empty
mapred.vertica.output.table.drop	Whether to drop/truncate tables	No	FALSE
mapred.vertica.output.delimiter	Optional output format delimiter	No	Bell - 0x7
mapred.vertica.output.terminator	Optional output format terminator	No	Backspace - 0x8