Hive/HCatalog Streaming Ingest Functional Specification and Design (Phase 1)

- Eric Baldeschwieler, Alan Gates, Roshan Naik
April 29th 2013

Introduction

Many systems provide a pipeline of records collected from one or more sources that are destined to be stored in Hadoop (e.g. Flume). Users want these records to be stored in Hadoop and available to be queried by Hive, Pig, or MapReduce with very low latency. They also want these records to be stored in the native format for the table the records are inserted to. This document proposes an interface and new functionality in HCatalog to meet these requirements. It currently only proposes the first installment of this system.

Definitions

Streaming Record Provider (SRP) - A system that is collecting records from one or more sources and inserting them into Hadoop, possibly in parallel. Examples include Flume or a connector to a message queue tool such as ActiveMQ.

Hadoop tools - Pig, Hive, MapReduce, webhcat, and any other tools that may attempt to access data inside a Hive table.

Chunk - A collection of records written by an SRP. These will be written and committed to HDFS. Once a chunk is committed to HDFS the SRP can consider the records to be safely stored and may remove them from its system.

Use Case

Actors:
- HCatalog
- A thread or process in an SRP. A single SRP may have many of these threads or processes. In this use case this will be referred to as the SRP process, though it may actually be a thread.
- Oozie

1. The SRP process opens a chunk via a web services call to write records to a table. It is returned a URL to which it can post records.
2. The SRP process streams records into the provided file URL via webhdfs. It is assumed that this will be for a short time period (<= 1m) since SRPs usually need to move records quickly. However, HCatalog imposes no requirement that this chunk transaction is short lived.
3. Commit or abort:
    a. The SRP process commits the chunk it has written by making another web services call. This only commits the records written by this process in this chunk.

Chunks being written simultaneously by other SRP processes will not be committed by this operation. Committed chunks are not yet readable by Hadoop tools.

b Alternatively the SRP aborts the chunk it has written via a web services call. This aborts only the records written to this chunk by this process. Chunks being written simultaneously by other SRP processes will not be aborted by this operation.

4 Oozie at a regular interval creates a new partition in the table from the currently committed chunks. Once this has completed the data will be available to Hadoop tools.

Requirements

1 HCatalog must provide a set of web services calls that allows the SRP to write these chunks in short time periods (<= 60 seconds). These systems do not have the megabytes of buffer space to store records that would be required to wait for more traditional Hadoop partition time frames (5-60 minutes).

2 For this release, the SRP must write records that match the current schema, line delimiter, column delimiter, and file format of the table. As part of opening the chunk the SRP will provide this information to the web service and the call will fail if those parameters do not match the table's current configuration.

3 Tables to be written to in this manner must be partitioned by one and only one column. It is assumed this column will contain a datestamp and or timestamp.

4 An Oozie job will be provided that, at a configurable interval, commits the files written by this interface into HCatalog. This must be done in a way that writers streaming data into the table are not disrupted and that any transactions not yet committed are not committed to the partition after it is committed. The frequency with which the commits are done must be configurable and at a minimum provide the option to commit at 5 minute, 15 minute, and 1 hour intervals.

Streaming Web Services Interface

The streaming interface will use Kerberos authentication as the rest of webhcat does.

Obtain a Chunk

This will be called by the SRP to get the file name into which a chunk of records will be written.
Verb: POST
URL: `.../webhcat/v2/streaming/chunkget`
POST Parameters:

Parameter	Description	Required	Default Value
user.name	User to act as	yes	
database	The name of the database the table to stream data into is in	no	“default” (the default database)
table	The name of the table to stream data into	yes	

schema	A description of the columns in the data and their data types. The format of the schema will match the JSON format returned by the ddl GET call on the table, with column names optional	yes	
format	How the data will be written; initially the only valid value will be CSV	yes	
record_separator	How records are delimited in the data; initially the only valid value is \n	yes, when format is CSV	
field_separator	How fields are delimited in a record; any single ASCII character is valid	yes, when format is CSV	

Results:

Return Code	**Indication/Contents**
200 Success	A JSON document that contains the webhdfs URL to write to
401 Unauthorized	The user is not authorized to write to the table indicated
404 Not Found	The database.table indicated does not exist
405 Method Not Allowed	The table indicated is not configured for accepting streaming input
415 Unsupported Media Type	The table information such as schema, format, record_separator, or field_separator provided by the SRP does not match those of the table

Example:
The SRP sends a POST to
`http://hadoop.acme.com/webhcat/v2/streaming/open?user.name=gates` with parameters:

```
table : logs
schema : {"columns" : [ { "type" : "string" }, { "type" : "string}] }
format : CSV
record_separator : '\n'
field_separator : ','
```

It receives back a status code 200 and a JSON document

```
{
   "chunkid" : "193",
   "chunkfileurl" : "http://xyz.com:50030/webhdfs/v1/tmp/hive-
streaming/default/logs/chunk_193",
```

```
   "chunkfile" : "/tmp/hive/data/streaming/default/logs/chunk_193"
}
```

Commit a Chunk

This will be called by the SRP when it has successfully finished writing a chunk.
Verb: POST
URL: `.../webhcat/v2/streaming/chunkcommit`
POST Parameters:

Parameter	Description	Required	Default Value
user.name	User to act as	only in insecure mode	
database	Name of the database into which the chunk being committed	yes	
table	Name of the table into which the chunk is being committed	yes	
chunkfile	The chunkfile returned by /chunkget	yes	

Results:

Return Code	Indication/Contents
200 Success	A Json document noting the new location of the committed chunk. `{ "chunkfile" : "....path.." }`
401 Unauthorized	The user is not authorized to write to the table indicated
404 Not Found	The chunk indicated does not exist

h3. Abort a Chunk

This will be called by the SRP when it has determined that the chunk should be abandoned.

Verb: POST
URL: `.../webhcat/v2/streaming/chunkabort`
POST Parameters:

Parameter	Description	Required	Default Value
user.name	User to act as	only in insecure mode	
chunkfile	chunkfile returned by /chunkget	yes	

Results:

Return Code	Indication/Contents
200 Success	
401 Unauthorized	The user is not authorized to write to the table indicated
404 Not Found	The chunk indicated does not exist

h3. Commit a Streaming Partition and Open a New One

This will be called by Oozie to commit an existing streaming partition and open a new streaming partition where chunks can be written. Once this is called, no new chunks will be committed to the old streaming partition.

Verb: POST
URL: `.../webhcat/v2/streaming/partitionroll`
POST Parameters:

Parameter	Description	Required	Default Value
user.name	User to act as	only in insecure mode	
database	The name of the database the table to open a streaming partition in is in	no	“default” (the default database)
table	The name of the table to open a streaming partition in	yes	
partition_column	Name of the partition column for this table	yes	
partition_value	Value to assign to the partition column for the new streaming partition that is being created	yes	

Results:

Return Code	Indication/Contents
200 Success	A JSON document with two a status result on the commit of the previous streaming partition and a URL for the directory of the new streaming partition. The commit status contains three fields, an integer status code returned by the "alter table add partition" command, any message returned by that command, and the directory that was used for the old streaming partition. The message and old streaming partition are set only if the commit fails.
401 Unauthorized	The user is not authorized to write to the table indicated
404 Not Found	The database.table indicated does not exist

415 Unsupported Media Type	The indicated partition_column does not match the partitioning column indicated in the metastore.

Example:
Oozie sends a POST to
`http://hadoop.acme.com/webhcat/v2/streaming/partitionroll?user.name=oozie` with parameters:
`table="logs", partition_column="ds", partition_value="201303070900"`

It receives back a status code 200 and a JSON document:

```
  {
        "commit_status" : {
            "status" : 0,
            "message" : null
            "old_streaming_partition" : null
        },
        "new_streaming_partition" :
"hdfs://nn.hadoop.acme.com:50030/hive/data/streaming/default/logs/ds=20130307
0900/"
    }
```

In the case where this is the first call on the table to open a new streaming partition then the commit_status structure will be null.

Example Work Flow

Oozie opens a streaming partition for a table that had not previously been configured for use with streaming ingest by calling
`http://webhcat.hadoop.acme.com/webhcat/v2/streaming/partitionroll?user.name=oozie` with POST parameters `table="logs", partition_column="ds", partition_value="201303091130".` In return it receives a JSON document:

```
  {
        "commit_status" : null,
        "new_streaming_partition" :
"http://nn.hadoop.acme.com:50030/hive/data/streaming/default/logs/ds=20130309
1130/"
    }
```

Assume that the table has a schema of (userid string, url string, clicktime timestamp).

One or more Flume agents then begin streaming records into the table's streaming partition. Each agent begins by calling
`http://hadoop.acme.com/webhcat/v2/streaming/chunkget?user.name=flume` with POST parameters:

```
table="logs"
schema='{"columns" : [ { "type" : "string" }, { "type" : "string" },
{ "type" : "timestamp" } ] }'
```

format="CSV", record_separator="\n"
field_separator="\0001".
This will return a JSON document like { "url" :
"http://nn.hadoop.acme.com:50030/webhdfs/v1/hive/data/streaming/default/logs/
ds=201303091130/chunk_1" }

The Flume agent will then use webhdfs to open and write data into that file by sending a PUT to http://nn.hadoop.acme.come:50030/webhdfs/v1/hive/data/streaming/default/logs/1/chunk_1?op=CREATE with no data, which will result in it receiving a redirect to a data node, which it will then again send a PUT to with the data to be loaded into the file. See the webhdfs documentation for create for more information on using that call.

Once the agent has finished writing the file it will call http://webhcat.hadoop.acme.com/webhcat/v2/streaming/commit?user.name=flume with POST parameters:
chunk="http://nn.hadoop.acme.com:50030/webhdfs/v1/hive/data/streaming/default/logs/ds=201303091130/chunk_1"

After fifteen minutes Oozie again calls http://webhcat.hadoop.acme.com/webhcat/v2/streaming/partitionroll?user.name=oozie with POST parameters table="logs", partition_column="ds", partition_value="201303091145". In return it receives a JSON document:

```
{
        "commit_status" : {
            "status" : 0,
            "message" : null,
            "old_streaming_partition" :
"http://nn.hadoop.acme.com:50030/hive/data/streaming/default/logs/ds=201303091130/"
        },
        "new_streaming_partition" :
"http://nn.hadoop.acme.com:50030/hive/data/streaming/default/logs/ds=201303091145/"
}
```

Design

Changes are required to the metastore database. The four new web services calls specified above will also be added to webhcat. New configuration parameters will also be required. An Oozie workflow will also need to be created that the user can parameterize and configure to use against their table.

Database Changes

Following new fields in the TBLS table will added to the database with the schema:

```
        CURRENT_STREAMING_PARTITION_PATH varchar(255),
```

```
CURRENT_STREAMING_PARTITION_VAL  varchar(255),
STREAMING_TMP_DIR                varchar(255),
CHUNK_ID                         int(11)
```

First two fields capture the path and value of the current streaming partition. `STREAMING_TMP_DIR` is the location for the uncommitted chunks. The `CHUNK_ID` field is used to issue unique IDs to each new chunk.

Additional Hive Settings

hive.streaming.tempdir : The default base directory to use for uncommitted streaming writes. If unspecified, it defaults to `/user/hive/warehouse/streaming/`. This setting will be used to determine the `STREAMING_TMP_DIR` value, unless admin overrides the value manually in the metastore.

hive.streaming.basedir : The directory in which the committed chunks are stored for the streaming partition. This location is used to determine the `CURRENT_STREAMING_PARTITION` value.

Webhcat Changes

Roll Partition

When webhcat receives a POST to `webhcat/v2/streaming/partitionroll` it will:

1. Check that the indicated table exists in the indicated database, if not return 404.
2. Call the authorization provider to ensure the indicated user is authorized to create partitions on this table. If not, return 401.
3. Check that the indicated partition column matches the partition column for the table. If not, return 415.
4. Create a directory for the new streaming partition. The name will be constructed using hive.streaming.basedir + dbname + tablename + partition_column + '=' + partition_value.
5. Update the STREAMING_INGEST table to set the CURRENT_STREAMING_PARTITION to the newly created directory.
6. Commit the previous streaming partition. This is simply an 'add partition' operation.
7. Return the results of the commit operation and the new streaming partition directory to the caller.

NOTE: There is a data loss scenario here, if the webhcat servlet dies between updating the STREAMING_INGEST table and committing the previous streaming partition. It may be possible to perform steps 5 and 6 in a single Metastore transaction.

Open

When webhcat receives a POST to `webhcat/v2/streaming/chunkget` it will:

1. Check that the indicated table exists in the indicated database, if not return 404.
2. Call the authorization provider to ensure the indicated user is authorized to create partitions on this table. If not, return 401.

3 Check that the indicated table has an entry in the STREAMING_INGEST table, if not return 405.
4 Check that the schema, format, record_separator, and field_separator indicated by the caller match what is recorded in the metastore, if not return 415.
5 Read the STREAMING_INGEST table for the indicated database and table, obtain the STREAMING_TMP_DIR value, and read and increment the NEXT_CHUNK_SEQUENCE atomically.
6 Construct a file name for the chunk using namenode + "/webhdfs/v1/" + STREAMING_TMP_DIR + NEXT_CHUNK_SEQUENCE
7 Return a JSON document with the constructed filename.

Commit

When webhcat receives a POST to `webhcat/v2/streaming/chunkcommit` it will:

1 Call the authorization provider to ensure the indicated user is authorized to write to this table. If not, return 401.
2 Check that the indicated chunk exists, if not return 404.
3 Read STREAMING_INGEST.CURRENT_STREAMING_PARTITION and move the indicated chunk file to the current streaming directory.
4 Return 200.

Abort

When webhcat receives a POST to `webhcat/v2/streaming/chunkabort` it will:

1 Call the authorization provider to ensure the indicated user is authorized to to write to this table. If not, return 401.
2 Check that the indicated chunk exists, if not return 404.
3 Remove the indicated chunk file from STREAMING_TMP_DIR.
4 Return 200.

Oozie Workflow

An Oozie workflow will be required that calls streaming/partitionroll. The table (and associated data such as partition column and value) must be parameterizable. Also, it must be possible to configure how often this workflow is scheduled. At the minimum it must be possible to run this at a 5 minute, 15 minutes, and 1 hour frequencies.

Phased Development

The intention is to break down the implementation into two phases in order to get some early experience with a simple system and incorporate feedback into the final deliverable. The breakdown of the two phases is as follows:

Phase 1: This document describes the design of phase 1 only.
This includes:

1 Enhancements to metastore to support streaming partitions.
2 Webhcat based API to get, commit and abort chunks.
3 Webhcat based API to support rolling the streaming partition.

This does not include:

1. DDL support to enable streaming
2. An active agent of any kind that will periodically trigger the rolling of streaming partitions. A separate Oozie job needs to be configured to periodically trigger partition rolling.
3. Ability to optimize, compact or do other processing of committed chunks at the time of rolling.
4. Ability to query the streaming partition before its rolled.

Phase 2:

In this phase we plan to addressing the following aspects:

1. DDL support to enable streaming.
2. Ability to query the streaming partition
3. Making HCat the single point of configuration so that a separate Oozie does not need to be configured manually for partition rolling. The rolling frequency is determined by the value set by the admin on the table.
4. The extent to which optimizing partition at roll time is supported. And how it would be supported.