StreamingUDFs

Streaming UDFs Proposal

Introduction

This document provides a proposal to add Streaming UDFs to Pig. The goal of Streaming UDFs is to allow users to easily write UDFs in scripting languages with no JVM implementation or a limited JVM implementation. Examples include:

- Languages without (widely-used) JVM implementations: **R, perl, ...**
- Languages with JVM implementations, but w/o support for C-extension modules
  - **python**: the jython UDFs from PIG-928 cannot make use of packages like nltk, numpy, and scipy that use C extensions Python extensions.
  - **ruby**: Some modules use C extensions, for which experimental JRuby support exists, but it's still experimental

Currently, the best way to use JVM-unfriendly code in these languages from Pig is the STREAM operator. STREAM has several limitations, however:

- **Boilerplate code**: Every streaming script needs the same boilerplate code to load the input, parse it into fields, cast those fields to types, and then repeat the process on the way out in reverse. This code is repeated across many scripts.
- **No type information**: Streaming scripts don't have access to pig types, so all type-casting has to be repeated in every streaming script
- **Extra unwanted fields**: Since stream operates on a relation rather than on fields, every streaming script has to deal with ALL fields, not just the subset of fields it needs. This results in unnecessary I/O, as well as increased brittleness, as streaming scripts have to be aware of any schema change, even for fields they don't need.

When these factors come together, it makes it much harder to write, execute, and maintain code from these languages.

Proposed Solution

The proposed solution is to add support for streaming UDFs that can be written in any language and run from within FOREACH statements, as jython UDFs currently can.

Using CPython as an example, users would be able to write a python UDF such as:

```
# import a library that requires CPython
import nltk

def part_of_speech(words):
    # tokenize to [word]
    tokens = nltk.word_tokenize(words)

    # tag with part of speech
    # produces [(word, part_of_speech)]
    tokens_with_pos = nltk.pos_tag(tokens)

    return tokens_with_pos
```

and then reference it from their Pig script like:

```
-- syntax for define needs work to disambiguate from embedded JVM UDFs
define my_streaming_udfs `my_streaming_udfs.py` ship('mystreamingudfs.py');

-- load up the data
searches = LOAD 'excite-small.log' USING PigStorage('\t') AS (user_id:chararray, timestamp:chararray, query:
chararray);

-- execute streaming UDF on subset of fields
with_parts_of_speech = FOREACH searches
                       GENERATE user_id, timestamp,
                                FLATTEN(my_streaming_udfs.part_of_speech(query));
```

I think this functionality could be built on top of the existing pig STREAM support. The basic idea is to provide the boilerplate code in each supported target language, and execute that in a streaming fashion. I think we'd need a few things in place:

Controller Script (Language-Specific)

We'd need a per-language controller script that pig would ship to the cluster and get invoked once per UDF. It would be written in the language to be supported, and would:

- Be launched in a separate process by Pig
- Dynamically import the user's UDF script(s)
- Pull in metadata about which UDF to call and how to call it (script, function name, expected fields, data types). This would likely be provided by pig in a separate metadata file.
- Optionally determine the expected output fields/data types from the UDF itself (TBD)
- Open up the input stream of data on stdin
- For each incoming tuple: pull the row, deserialize, type cast, and pass to the UDF
- Receive the output from the UDF, serialize and pipe back to stdout

DEFINE for StreamingUDF

We'd want to either update DEFINE or add a new command to support streaming UDFs. The syntax could be very similar to the existing DEFINE for STREAMING, but would need a way to specify what language of script was being called (allowing it to choose which Controller Script to use)

StreamingUDF EvalFunc

We'd need a new EvalFunc for running streaming UDFs. It would use much of the functionality of the STREAM operator, but additionally would:

- Ship the per-language Controller Script into the cluster along with user's UDF script
- Write out and pass metadata about the UDF to be called to the Controller Script
- Exec the Controller Script (TBD how this is done, see Open Questions below)
- Be able to receive a spec for the UDF's output schema from the controller script along with the output data itself

Open Questions / Thoughts

A few things to be figured out:

- How can we return the output type information back to pig? Perhaps we could support something like the @outputSchema decorator in python at least, and have the controller script gather that information and provide it back to pig in a separate file?
- How do we execute the stream process? Particularly, how do we know where the scripting language executable lives? We could get this from the UDF script and transfer it to the controller script, defaulting to #!/usr/bin/env python.
- How can we ensure that this process will work for a good cross-section of scripting languages to support?