ISIT312 Big Data Management

Apache Pig

Dr Fenghui Ren

School of Computing and Information Technology -
University of Wollongong

Apache Pig

Outline

A short history of Pig

Pig started out as a research project in Yahoo! Research, where Yahoo! scientists designed it and produced an initial implementation

As explained in a paper presented at SIGMOD in 2008, the researchers felt that the MapReduce paradigm presented by Hadoop is too low-level and rigid, and leads to a great deal of custom user code that is hard to maintain and reuse

At the same time they observed that many MapReduce users were not comfortable with declarative languages such as SQL

Thus they set out to produce a new language called Pig Latin that we have designed to fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of MapReduce

Yahoo! Hadoop users started to adopt Pig and a team of development engineers was assembled to take the research prototype and build it into a production-quality product

About this same time, in fall 2007, the first Pig release came a year later in September 2008. Later that same year, Pig graduated from the Incubator and became a subproject of Apache Hadoop

What is Pig ?

Pig provides an engine for processing data flows in parallel on Hadoop

It includes a language Pig Latin, for representation of the data flows

Pig Latin includes operators for many of the traditional data operations (join, sort, filter, etc.), as well as the ability for users to develop their own functions for reading, processing, and writing data

Pig runs on Hadoop, in the sense that it makes use of both HDFS, and is built on top of Hadoop processing framework MapReduce

Data flow language

The language of Pig, called Pig Latin is a dataflow language

It allows the users to describe how data from one or more inputs should be read, processed, and then stored to one or more outputs in parallel

The data flow in Pig is represented by a Directed Acyclic Graph (DAG)

There is no if statement or for loops in the Pig Latin

Comparing query and data flow languages:

- SQL is oriented around answering one question, while Pig specifies how data are manipulated
- SQL is oriented around answering one query, while Pig is designed for sequences of operations

Pig Latin

WordCount example in Pig Latin

WordCount example

```
input = load 'file' as (line);
-- Load input from the file named Mary, and call the single field in the record
   'line'

words = foreach input generate flatten(TOKENIZE(line)) as word;
-- TOKENIZE splits the line into a field for each word
-- flatten takes the collection of records returned by TOKENIZE and produce
   a separate record for each one, calling the single field in the record word

grpd = group words by word;
-- Now group them together by each word

cntd = foreach grpd generate group, COUNT(words);
-- Count them

dump cntd;
-- Print out the results
```

Differences between Pig and MapReduce

MapReduce programming model is expressive but relatively low-level, while Pig can be seen as a linguistic abstraction on MapReduce

- For example, the implementation of standard data-processing operations (such as join, filter, group by, etc.) is non-trivial in MapReduce, but these operations are primitive in Pig Latin

Pig Latin reduces the development time of MapReduce programming and is much lower cost to write and maintain than Java code for MapReduce

On the other hand, Pig does not support the object-oriented development

Also, the execution of Java programs for MapReduce is usually more efficient than an equivalent Pig script

Apache Pig

Outline

Grunt: Interactive shell

Grunt enables users to enter Pig Latin interactively and provides a shell for users to interact with HDFS

To enter Grunt, type

Starting Grunt

```
pig
```

will result in the prompt

Grunt prompt

```
grunt >
```

This will interact with a cluster configuration set in the classpath of Pig; the following command will gives a shell ability to interact with a local file system

Starting Grunt

```
pig -x local
```

Entering Pig Latin scripts in Grunt

Pig Latin commands can be entered directly into Grunt. For example:

Pig Lagtin commands

```
pig
grunt> dividends = load "NYSE_dividends" as (exchange, symbol, data,dividend)
grunt> symbols = foreach dividends generate symbl;
... Error during parsing. Invalid alias: symbl ...
grunt> symbols = foreach dividends generate symbol;
grunt> dump symbols;
```

Pig will not start executing the Pig Latin you enter until it sees either a `store` or `dump`

However, it will do basic syntax and semantic checking to help you catch errors quickly

To quit Grunt, type

Quiting Grunt

```
grunt> quit
```

HDFS commands in Grunt

Besides entering Pig Latin interactively, the other major use of Grunt is to act as a shell for HDFS

In versions 0.5 and later of Pig, all hadoop `fs` shell commands are available

For example

Hadoop commands in gruint

```
grunt> fs -ls
```

lists all the files and folders at the current HDFS user home folder

Other commands include `cat`, `copyFromLocal`, `rm`, ...

Apache Pig

Outline

Types

Data types in Pig can be divided into two categories: scalar types, which contain a single value, and complex types, which contain other types

Scalar Types

`int`

- An integer; integers are represented in interfaces by `Java.lang.Integer` and stored a four-byte signed integer numbers

`long`

- A long integer; long integers are represented in interfaces by `java.lang.Long` and store an eight-byte signed integer numbers

`float`

- A floating-point number; floating point numnbers are represented in interfaces by `java.lang.Float` and use four bytes to store their values

Scalar Types

`double`

- A double-precision floating-point number; double precision floating point numnbers are represented in interfaces by `java.lang.Double` and use eight bytes to store their values

`chararray`

- A string or character array; string or character arrays are represented in interfaces by `java.lang.String`
- Constant chararrays are expressed as string literals with single quotes, for example, `'fred'`

`bytearray`

- A blob or array of bytes; blobs or arrays of bytes are represented in interfaces by a Java class `DataByteArray` that wraps a Java `byte[ ]`

Complex Types

`map`

A map in Pig is a `chararray` to data element mapping, where that element can be any Pig type, including a complex type.

The `chararray` is called a key and is used as an index to find the element, referred to as the value

By default there is no requirement that all values in a `map` must be of the same type

`map` constants are formed using brackets to delimit the map, a hash between keys and values, and a comma between key-value pairs

For example, `['name'#'bob', 'age'#55]` will create a map with two keys, `"name"` and `"age"`

Note that the first value is a `chararray`, and the second is an `int`

Complex Types

`tuple`

A `tuple` is a fixed-length, ordered collection of Pig data elements

`tuples` are divided into fields, with each field containing one data element

These elements can be of any type and they do not all need to be the same type

A `tuple` is analogous to a row in SQL, with the fields being SQL columns

Tuple constants use parentheses to indicate the tuple and commas to delimit fields in the tuple

For example, `('bob', 55)` describes a tuple constant with two fields

Complex Types

`bag`

A `bag` is an unordered collection of tuples

Because it has no order, it is not possible to reference tuples in a bag by position

Like tuples, a `bag` can, but is not required to, have a schema associated with it

In the case of a `bag`, the schema describes all tuples within the `bag`

`bag` constants are constructed using braces, with tuples in the `bag` separated by commas

For example, `{('bob', 55), ('sally', 52), ('john', 25)}` constructs a `bag` with three tuples, each with two fields

Null Data

Pig includes the concept of a data element being `null`

Data of any type can be `null`

In Pig, the concept of `null` is the same as in SQL, which is completely different from the concept of null in C, Java, Python, etc

In Pig a `null` data element means the value is unknown

Schemas

The easiest way to communicate the schema of your data to Pig is to explicitly tell Pig what it is when you load the data

Loading with a schema

```
dividends = load 'NYSE_dividends' as
  (exchange:chararray, symbol:chararray, date:chararray, dividend:float);
```

Pig now expects your data to have four fields and if it has more, it will truncate the extra ones

If it has less, it will pad the end of the record with `nulls`

It is also possible to specify the schema without giving explicit data types. In this case, the data type is assumed to be `bytearray`

Schema with no data types

```
dividends = load 'NYSE_dividends' as (exchange, symbol, date, dividend);
```

Schemas

You can declare data as complex data types in a schema:

Declaring complex data types

Data type	Syntax	Example
map	map[] or map[type], where type is any valid type This declares all values in the map to be of this type	as (a:map[], b:map[int])
tuple	tuple() or tuple(list_of_fields), where list_of_fields is a comma-separated list of field declarations	as (a:tuple(), b:tuple(x:int, y:int))
bag	tuple() or tuple(list_of_fields), where list_of_fields is a comma-separated list of field declarations	as (a:bag{}, b:bag{t:(x:int, y:int)})

Schema

We can get the schema of a relation by using the `describe` operation

Describing a schema

```
dividends = load 'NYSE_dividends' as
            (exchange:chararray, symbol:chararray, date:chararray, dividend:float);
described dividends;
grpd: {exchange: chararray, stock: chararray, date: chararray,dividends: float}
```

Question: what is the output of the following

Describing a schema

```
dividends = load 'NYSE_dividends' as (exchange, symbol, date, dividend);
described dividends;
```

References

Gates A., Programming Pig, O'Reilly Media, Inc., 2011, (Available in `READINGS` folder)

Vaddeman B., Beginning Apache Pig: Big Data Processing Made Easy, Apress 2016 (Available in `READINGS` folder)