Pig

CMPSC/DS 410

Pig

Learning Objectives

- Understand the Pig programming language
- Understand how features of Pig correspond to Map Reduce
- Be able to develop MapReduce software applications using Pig

Reading

- "Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience" by Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, Utkarsh Srivastava

How to Develop a MapReduce Application in Hadoop?

- Using Pig: A high level platform for developing MapReduce programs
- High-level language
  - Make key programming constructs easy to implement
  - Pig Latin: The language
  - Pig shell: grunt (executable is pig)

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int);

B = FILTER A by retweet > 20;

C = GROUP B by tweeter;

D = FOREACH C GENERATE group AS tweeter, SUM (B.retweet);

STORE D INTO ‘high-retweet-counts’;
```

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int);

B = FILTER A by retweet > 20;

C = GROUP B by tweeter;

D = FOREACH C GENERATE group AS tweeter, SUM (B.retweet);

STORE D INTO ‘high-retweet-counts’;
```

- Map operations
  - LOAD ... AS
  - FILTER
- Shuffle/Sort operations
  - GROUP ... BY
- REDUCE operations
  - SUM

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int);

B = FILTER A by retweet > 20;

C = GROUP B by tweeter;

D = FOREACH C GENERATE group AS tweeter, SUM (B.retweet);

STORE D INTO ‘high-retweet-counts’;
```


- Pig is a dataflow language, inspired by SQL.
- Pig Latin is **typed**
  - Relations have types
  - Do not overwrite relations
  - Use `describe A;` to get type of a relation
- Evaluation is **lazy**
  - No execution is performed until needed
  - Allows Pig to plan transformation to MapReduce
  - STORE (save as file)
  - DUMP (dump to screen)
- End command with `;`

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int); describe A;
```

A: {tweeter: bytearray,text: bytearray,retweet: int}

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int); describe A;
```

A: {tweeter: bytearray,text: bytearray,retweet: int}

```
B = FILTER A by retweet > 20; describe B;
```

B: {tweeter: bytearray,text: bytearray,retweet: int}

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int); describe A;
```

A: {tweeter: bytearray,text: bytearray,retweet: int}

```
B = FILTER A by retweet > 20; describe B;
```


B: {tweeter: bytearray,text: bytearray,retweet: int}

```
C = GROUP B by tweeter; describe C;
```


C: {group: bytearray,B: {(tweeter: bytearray,text: bytearray,retweet: int)}}

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int); describe A;
```

A: {tweeter: bytearray,text: bytearray,retweet: int}

```
B = FILTER A by retweet > 20; describe B;
```

B: {tweeter: bytearray,text: bytearray,retweet: int}


```
C = GROUP B by tweeter; describe C;
```

C: {group: bytearray,B: {(tweeter: bytearray,text: bytearray,retweet: int)}}


```
D = FOREACH  C GENERATE  group AS tweeter, SUM (B.retweet); describe D;
```

D: {tweeter: bytearray,long}

An Example of Pig Latin

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int); describe A;
```

A: {tweeter: bytearray,text: bytearray,retweet: int}

```
B = FILTER A by retweet > 20; describe B;
```


B: {tweeter: bytearray,text: bytearray,retweet: int}

```
C = GROUP B by tweeter; describe C;
```


C: {group: bytearray,B: {(tweeter: bytearray,text: bytearray,retweet: int)}}


```
D = FOREACH  C GENERATE  group AS tweeter, SUM (B.retweet); describe D;
```

D: {tweeter: bytearray,long}

An Example of Pig Latin

```
STORE D INTO ‘high-retweet-counts’;
```

- Throws error because we don't have 'mytweets file'
- Feature/benefit of lazy evaluation
  - Can test out script and typecheck it
  - Pig can optimize it
  - Finds runtime errors later
- Documentation:
  - http://pig.apache.org/docs/r0.16.0/basic.html
  - Our version is 0.16
  - Latest is 0.17

Representing Data in Pig

- Scalar Types
  - int
  - long
  - double
  - datetime
  - chararray
  - bytearray
- Complex Types
  - Tuple: an ordered list
  - Map: a set of key value pairs
  - Bag: an unordered collection of tuples

```
C = GROUP B by tweeter; describe C;
```

C: {group: bytearray,B: {(tweeter: bytearray,text: bytearray,retweet: int)}}

- C is *outer* bag (Relation)
- C.B is *inner* Bag
- C.B is a set of tuples
  - fields: tweeter, text, retweet

Type declaration

- AS clause in LOAD, STREAM, FOREACH operators

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet:int);

D = FOREACH  C GENERATE  group AS tweeter, SUM (B.retweet) as total;
```


- Can define both the name and the type of a column
- If the type of a column is not explicitly defined, its type (by default) is bytearray


```
describe A;
```


A: {tweeter: bytearray,text: bytearray,retweet: int}

How to load data into Pig

- using field-delimited text format (PigStorage)
- using binary files (BinStorage)
- TextLoader: data from a plain-text format
- Ex:

```
A = LOAD ‘student’ USING PigStorage(‘,’) AS (twitter: chararray, text: chararry, retweet: int);
```

  - A is relation name, technically an outer bag
  - Specify delimiter in PigStorage()
- Dump to screen

```
DUMP A;
```

  - Don't do this for big data.
  - Only use DUMP for testing

Bag in Pig vs Relation in SQL

- A bag is an unordered collection of typed or untyped tuples
  - Type is bytearray if none given
  - Tuples can have different lengths
  - A position in tuples of a bag can contain data of different types.
    - i.e. if type of tuple field is bytearray, you can store ints, strings, etc.
  - Tuples in bag are not ordered.
  - Recursion: a bag can store a tuple that has a field that is a bag
- A relation is a collection of (typed) tuples
  - A position in tuples of a relation contains data of only one type.
  - Tuples are ordered if indexed by "key".

Map

- [key#value, key#value, ...]
- Key must be chararray data type
- Key must be unique
- Key can be used to represent “column name”

LOAD in Pig Supports MapReduce

- The syntax of a LOAD statement specifies the data element (a tuple)
- The “semantics” of a LOAD statement generates an aggregate of tuples (i.e., bags), whose order is irrelevant.
- Traditional READ/LOAD statement applies to one input source (e.g., file), where the order of the data in the source determines the order the data element gets processed.
- In Pig, data source can be
  - file
  - directory of files
  - paths with wildcards (e.g., "words*)

Referencing a column by position

- Accessing fields in tuples
  - $0: The first column in a tuple
  - $1: The second column in a tuple

```
A=LOAD ‘mytweets’ AS (tweeter, text, retweet);
B = FILTER A by $2 >20;
```


- Do not do this
  - Better to reference a column by name than by position.
  - More readable
  - Only use when field has no name

```
D = FOREACH  C GENERATE  group AS tweeter, SUM (B.retweet); describe D;
```

D: {tweeter: bytearray,long}

  - Even better: give the field a name

```
D = FOREACH  C GENERATE  group AS tweeter, SUM (B.retweet) as total;
```

Group by

```
A = LOAD ‘student’ USING  PigStorage() AS (tweeter: chararray, text: chararry, retweet: int);
C = GROUP A by tweeter;
DUMP C;
```

```
(BigTen,   {(BigTen, ....., 3),
            (BigTen, ...., 20),
            (BigTen, ....., 7)  }  )
(gopsu,    {(gopsu, ....., 120),
            (gopsu, ....,  78)  }  )
```



```
describe C;
C: {group: chararray,A: {(tweeter: chararray,text: chararray,retweet: int)}}
```

- group field: "BigTen"
  - Note it is not called **tweeter**
- A field: {(BigTen, ..., 3), (BigTen, ..., 20), (BigTen, ..., 7) }
  - Note that A.tweeter is the same as group

Group in Pig supports MapReduce

- Can implement aggregation (shuffle/sort) step between the Map step and the Reduce step.
- Regroup the intermediate results of the map step to prepare for the reduce step.
- The output of Group is (group_name, bag) where the bag contains all tuples with the key:

```
(BigTen,   {(BigTen, ....., 3),
            (BigTen, ...., 20),
            (BigTen, ....., 7)  }  )
(gopsu,    {(gopsu, ....., 120),
            (gopsu, ....,  78)  }  )
```


- Naturally maps to (key,value) pairs:
  - key = group_name
  - value = the bag

FOREACH ..... GENERATE ...

- Applies to each tuple of a bag
- Generates a new bag
- This feature supports MapReduce
  - Implement the Mapping function or the Reduce function


```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet: int);
C = GROUP A by tweeter;
D = FOREACH  C  GENERATE group as tweeter, SUM (A.retweet) as total;
```


- Does this implement a mapping function or a reduce function?

FOREACH ..... GENERATE ...

- Applies to each tuple of a bag
- Generates a new bag
- This feature supports MapReduce
  - Implement the Mapping function or the Reduce function


```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet: int);
C = GROUP A by tweeter;
D = FOREACH  C  GENERATE group as tweeter, SUM (A.retweet) as total;
```


- Does this implement a mapping function or a reduce function?
  - Mapper: to output (tweeter, retweet)
  - At reducer this becomes: (tweeter, retweet_list)
  - Reducer: sums the retweets

A side note on conversion to MapReduce

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet: int);
B = FILTER A by text MATCHES '.*psu.*';
C = GROUP B by tweeter;
D = FOREACH  C  GENERATE group as tweeter, SUM (B.retweet) as total;
E = FILTER D by total > 30;
```

A side note on conversion to MapReduce

```
A = LOAD ‘mytweets’ AS (tweeter, text, retweet: int);
B = FILTER A by text MATCHES '.*psu.*';
C = GROUP B by tweeter;
D = FOREACH  C  GENERATE group as tweeter, SUM (B.retweet) as total;
E = FILTER D by total > 30;
```


- Mapper
  - Parses line to get tweeter, text, and retweet (A)
  - Converts retweet to int (A)
  - Checks if text contains **psu** using regex matching (B)
  - Outputs (key, value) where key = tweeter, value = retweet (C, D)
    - Why? Other fields are not needed for this piggob. (D)
- Reducer:
  - inputs key=tweeter, value-list=retweet_list
  - sums up retweets (D)
  - Checks if this sum is > 30 (E)
    - if yes, outputs key=tweeter, value=sum of retweets
    - if no, produces no output for this key, value-list pair

COGROUP: A GROUP with a join

```
A = LOAD 'data1' AS (owner:chararray,pet:chararray);
B = LOAD 'data2' AS (friend1:chararray,friend2:chararray);
```

```
DUMP A;
(Alice,turtle)
(Alice,goldfish)
(Alice,cat)
(Bob,dog)
(Bob,cat)
```

```
DUMP B;
(Cindy,Alice)
(Mark,Alice)
(Paul,Bob)
(Paul,Jane)
```


http://pig.apache.org/docs/r0.16.0/basic.html


```
X = COGROUP A BY owner, B BY friend2;
DESCRIBE X;
X: {group: chararray,A: {owner: chararray,pet: chararray},
    B: {friend1: chararray,friend2: chararray}}
DUMP X;
(Alice,  {(Alice,turtle),(Alice,goldfish),(Alice,cat)},  {(Cindy,Alice),(Mark,Alice)})
(Bob,  {(Bob,dog),(Bob,cat)},    {(Paul,Bob)})
(Jane,  {},    {(Paul,Jane)})
```

Comments in Pig

```
/* A multiline Comments
looks like this */

DUMP B; -- single line comment
```

What if we want to define our own mapping function?

- Use User Defined Function (UDF) in Pig
- Example:
  - Suppose we want to classify tweets by their sentiment (e.g. positive vs negative)


```
REGISTER sentiment.jar; -- java file containing UDF
A = LOAD ‘tweets’ AS (tweeter:chararray, text:chararray);
B = FOREACH A GENERATE tweeter, text,sentiment.classifymood(text); --use the UDF
```


- UDFS can be written in
  - java
  - Pythoh
  - Javascript