Pig

Data at Scale

Pig

- Control Language for Map Reduce on top of Hadoop
- Consists of:
    - Language to express data flows: Pig Latin
    - Execution program to run Pig Latin programs
        - Local execution in a single JVM
        - Distributed execution on a Hadoop cluster

Pig

- Map-reduce has long development cycles:
  - Writing mappers and reducers
  - Compiling and packaging code
  - Submitting the jobs
  - Retrieving the results
- Pig can process terabytes of data with half a dozen lines of Pig Latin from the console

Pig

- Developed at Yahoo to allow employees and researchers to mine large Yahoo datasets
- Allows to introspect data structures
- Can perform sample runs on representative sample of data

Pig

- Written to be extensible
    - Loading, storing, filtering, grouping, joining can be altered by **User-Defined Functions** (UDF)

- Pig is often not as fast as pure map-reduce
    - But the distance is shrinking with each release

Pig

- Local mode:
    - Pig runs a single JVM and accesses the local file system
    - Is set by running pig with the -x or -exectype option
    - pig -x local   (local mode)
    - pig -x mapreduce   (map-reduce mode)

Pig

- Map-reduce mode
    - Pig translates queries into MapReduce jobs
    - Pig then runs them on a Hadoop cluster
    - Uses HADOOP-HOME environment variable for finding Hadoop client

Running Pig

- Three ways to execute pig programs
    - Script
        - `pig script.pig`
    - Grunt
        - Interactive shell
        - `pig -x mapreduce`
    - Embedded
        - Use `PigServer` class with `Pigrunner`

Running Pig

- Grunt
  - Has line editing commands
    - CTRL-P — previous command, CTRL-N — next command, CTRL-E — end of line
  - Has command completion for pig keywords invoked by TAB
- PigPen:
  - Eclipse plug-in development environment for developing pig programs

Running Pig

- Pig Latin and SQL
  - Pig Latin is a data-flow language
    - Specify the way to the output
  - SQL is a descriptive programming language
    - Specify the output
  - Pig Latin works on any source of tuples (Pig eat everything)
    - but can specify schemas
  - SQL needs to adhere to tables with schemas

Running Pig

- Pig Latin and SQL
    - Pig supports complex, nested data structures
        - and functional operators to change them
    - SQL has only simple data structures
    - Pig Latin has no indices and similar performance enhancing auxiliary data structures
    - SQL allows to define indices, etc to speed up queries

Pig Latin

- Pig Latin program is a collection of statements
- Each statement is an operation of command
- `grouped_records = GROUP records BY year;`
- Statements are usually terminated by a semicolon
  - Exception: statements for interactive use such as `ls /`
  - Statements with semicolon can be split over several lines
- Pig Latin uses
  - SQL-style comments - -
  - C-style comments /* */

Pig Latin

- LOAD — loads data from the file system

```
records = LOAD ‘file’ AS
 (year: int, temperature: int, quality: int);
```

- STORE — saves data to the file system

```
 STORE A INTO ‘output/b’ USING PigStorage(‘;’);
```

- DUMP — print relation to console

Pig Latin

- Filter: Use a Boolean condition

```
divs = LOAD('NYSE_dividends') AS
(exchange: chararray, symbol: chararray, data: chararray,
 dividends: float);

startswithcm = FILTER divs BY symbol MATCHES 'CM.*';

positive = FILTER divs BY NOT dividend == 0.0;
```

Pig Latin

- GROUP — collect records with the same key

```
divs = LOAD(‘NYSE_dividends’) AS
(exchange: chararray, symbol: chararray, date: chararray,
 dividends: float);

grpd = GROUP daily BY symbol, date;

grpd2 = GROUP daily BY (exchange, symbol);

grpd3 = GROUP daily BY ALL;

cnt = FOREACH grpd GENERATE GROUP, COUNT(daily);
```

Pig Latin

- Group:
  - Group creates relations
    - First field is the grouping field (called group)
    - Second field is a bag of grouped fields with the same schema as the original relation

Pig Latin

- FOREACH … GENERATE — removes rows from a relation

```
A = LOAD 'input/pig/foreach/A'
  AS (f0:chararray, f1:chararray, f2:int);
DUMP A;
B = FOREACH A GENERATE $0, $2+1, 'Constant';
DUMP B;
DESCRIBE B;
C = FOREACH A GENERATE $0, (int) $2 AS f1, 'Constant' AS f2;
DUMP C;
DESCRIBE C;
```

  - Example:
    - A: (Joe, cherry, 2)
    - B: (Joe, 3, Constant)
    - C: fields are renamed

Pig Latin

- JOIN: Inner join by common attribute

```
C = JOIN A BY  $0,  B BY $1;
```

- using ‘replicated’ for replicated join
    - First relation is the large one, the other ones are smaller

```
C = JOIN A BY $0, B BY $1 USING ‘replicated’;
```

- specifying outer joins

```
C = JOIN A BY $0 LEFT OUTER, B BY $1;
```

Pig Latin

- COGROUP
  - Allows nested set of output tuples
- CROSS
  - Creates cross-product

```
I = CROSS A,B;
```

Pig Latin

- Sorting data with ORDER

  -
    ```
    B = ORDER A BY $0, $1 DESC;
    ```

Pig Latin

- Combining data with UNION
-
  ```
  C = UNION A, B;
  ```

Pig Latin

- SPLIT

```
SPLIT records INTO good_records if temperature is not null,
bad_records if temperature is null
```

Pig Latin

- Diagnostics:
  - DESCRIBE — print schema
  - EXPLAIN — print logical and physical plan
  - ILLUSTRATE — show sample execution of the logical pan

Pig Latin

- Using UDF
  - REGISTER — register a JAR file with the Pig runtime
  - DEFINE — creates alias for macro, UDF, …
  - IMPORT — import macros defined in a separate file

Pig Latin

- Commands:
  - cat — print contents of a file
  - cd — change current directory
  - copyFromLocal — copy local file to Hadoop
  - copyToLocal — copy from Hadoop fs to local
  - cp — copy files
  - fs — access Hadoop file system
  - ls — list files
  - mkdir — create new directory
  - mv — move files
  - pwd — print current working directory path
  - rm — remove file

Pig Latin

- Expressions:
    - c.$1 c.name   projection
    - item#’Coat’ — value associated with key in a map
    - (int) f — casting
    - arithmetic: $2+$3, 5*$1+$2
    - conditional: x ? y : z
    - comparisons: $1<$2
    - Booleans: or, and, matches, is null, is not null
    - Flatten: removes a level of nesting from bags and tuples

Pig Latin

- Types:
    - int, long
    - float, double
    - chararray
    - bytearray
    - tuple: (1, ‘apple’)
    - bag { (1, ‘apple’), (2)}
    - map

Pig Latin

- Validation
  - Pig enforces constraints in a table at load time
  - Failure results in an offending value being made into a null

Pig Latin

- User - Defined Functions (UDF)

```
filtered_records = FILTER records BY temperature != 9999 AND
isGood(quality)
```

  - UDFs are subclasses of FilterFunc which is derived from EvalFunc
  - UDFs are compiled and packaged in a JAR file
  - Use Register to tell Pig about the JAR file

Pig Latin

- UDF
  - Can also use scripting languages

```
register ‘production.py’ using jython as bbccdd;
player = load ‘baseball’ as (name:chararray, team:
  chararray, pos: bag{t:(p:charrarray)}, bat:map[]);

nonnull = filter player by bat#’slugging_percentage’
is not null and bat#’on_base_percentage’ is not null;

calcprod = foreach nonull genererate name,
bbccdd.production(
 (float)bat#’slugging_percentage’,
 (float)bat#’on_base_percentage’);
```