Pig

Igor Yakushin
ivy2@uchicago.edu

Pig: introduction

- High level language - **Pig Latin**
- Compiler translates Pig Latin into MapReduce jobs
- It is a **dataflow language** where you define a data stream and a set of transformations applied to it.
- Operations: load, store, dump, filter, foreach, group, join, order by, distinct, limit, sample, etc.
- You can specify data types to help Pig to optimize a program or you can let it figure it out

Pig: how to run

Pig programs can run in three ways:

- as a script
  - on a local computer

    ```
    pig -x local milesPerCarrier.pig
    ```

  - on a cluster

    ```
    pig -x mapreduce milesPerCarrier.pig
    ```

- interactively using Grunt interpreter

  ```
  pig -x local
  ```

- Embedded in other languages such as Java, Python, JavaScript

Example 1

```
-- Reading records from a simple csv file,
-- where records are separated by ',', into a Pig relation.
-- If data type is not specified, Pig will figure it out.
-- If it is, it might improve performance

records = LOAD 'pig/test1.csv' USING PigStorage(',') AS (name, age:int, money:float);
-- Print schema
describe records;
-- Dump relation to screen. Pig is translated into MapReduce and job is launched.
dump records;

-- Project only some columns from a relation
projection = FOREACH records GENERATE name,money;
describe projection;
dump projection;

-- If we want to apply some aggregation to a column, records must be grouped first
mrecs = GROUP records ALL;
describe mrecs;
dump mrecs;

tot = FOREACH mrecs GENERATE SUM(records.money);
dump tot;

records1 = order records by age asc;
dump records1;

records2 = union records, records1;
dump records2;
```

Example 1

```
--Finding average money per age group and measuring the size of the age group

agegroups = GROUP records by age;
describe agegroups;
dump agegroups;

avgmoneyperage = FOREACH agegroups GENERATE group, AVG(records.money);
describe avgmoneyperage;
dump avgmoneyperage;

countagegroup = FOREACH agegroups GENERATE group, COUNT(records);
dump countagegroup;

--Select only 3 records
records3 = limit records 3;
dump records3;

--Filter records by some condition
records4 = filter records by age > 20;
dump records4;

-- Derive a new column
records5 = foreach records generate name, money/age as mpa;
describe records5;
dump records5;

-- Inner join by name
records6 = join records by name, records5 by name;
describe records6;
dump records6;
```

Example 2

```
in = load 'pig/mary.txt' as (line);
-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, calling the single
-- field in the record word.
words = foreach in generate flatten(TOKENIZE(line)) as word;
grpd = group words by word;
cntd = foreach grpd generate group, COUNT(words);
store cntd into 'cntd.out';
```