APACHE PIG

Why Pig?

Pig stands on top of mapreduce

- Writing mappers and reducers by hand takes a long time.
- Pig introduces *Pig Latin*, a scripting language that lets you use SQL-like syntax to define your map and reduce steps.
- Highly extensible with user-defined functions (UDF's)

manipulate data

MapReduce

TEZ

faster than mapreduce for take advantage of job independence then run them parallely

YARN where things are and how

HDFS garage

Running Pig

- Grunt
- Script
- Ambari / Hue

An example

- Find the oldest 5-star movies

```
ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS
      (userID:int, movieID:int, rating:int, ratingTime:int);
```

This creates a *relation* named “ratings” with a given *schema*.

(660,229,2,891406212)
(421,498,4,892241344)
(495,1091,4,888637503)
(806,421,4,882388897)
(676,538,4,892685437)
(721,262,3,877137285)

Use PigStorage if you need a different delimiter.

```
metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING
      PigStorage('|')AS (movieID:int, movieTitle:chararray,
      releaseDate:chararray, videoRelease:chararray,
      imdbLink:chararray);

DUMP metadata;
```

(1,Toy Story (1995),01-Jan-1995,,http://us.imdb.com/M/title-exact?Toy%20Story%20(1995))
(2,GoldenEye (1995),01-Jan-1995,,http://us.imdb.com/M/title-exact?GoldenEye%20(1995))
(3,Four Rooms (1995),01-Jan-1995,,http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995))
(4,Get Shorty (1995),01-Jan-1995,,http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995))
(5,Copycat (1995),01-Jan-1995,,http://us.imdb.com/M/title-exact?Copycat%20(1995))

Creating a relation from another relation; FOREACH / GENERATE

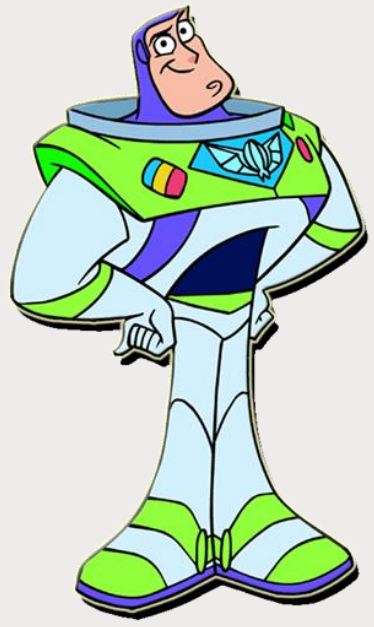
```
metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
      AS (movieID:int, movieTitle:chararray, releaseDate:chararray,
      videoRelease:chararray, imdbLink:chararray);

nameLookup = FOREACH metadata GENERATE movieID, movieTitle,
      ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) AS releaseTime;
```

(1,Toy Story (1995),01-Jan-1995,,http://us.imdb.com/M/title-exact?Toy%20Story%20(1995))

↓

(1,Toy Story (1995),788918400)



Group By

```
ratingsByMovie = GROUP ratings BY movieID;
DUMP ratingsByMovie;
```

(1,{(807,1,4,892528231),(554,1,3,876231938),(49,1,2,888068651), … }

(2,{(429,2,3,882387599),(551,2,2,892784780),(774,2,1,888557383), … }

ratingsByMovie: {group: int,ratings: {(userID: int,movieID: int,rating: int,ratingTime: int)}}

```
avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID,
      AVG(ratings.rating) AS avgRating;

DUMP avgRatings;
```

```
(1,3.878318584070796)
(2,3.2061068702290076)
(3,3.033333333333333)
(4,3.550239234449761)
(5,3.302325581395349)
```

```
DESCRIBE ratings;
DESCRIBE ratingsByMovie;
DESCRIBE avgRatings;
```

ratings: {userID: int,movieID: int,rating: int,ratingTime: int}

ratingsByMovie: {group: int,ratings: {(userID: int,movieID: int,rating: int,ratingTime: int)}}

avgRatings: {movieID: int,avgRating: double}

FILTER

```
fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;
```

(12,4.385767790262173)
(22,4.151515151515151)
(23,4.120879120879120 4)
(45,4.05)

JOIN

```
DESCRIBE fiveStarMovies;

DESCRIBE nameLookup;

fiveStarsWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;

DESCRIBE fiveStarsWithData;

DUMP fiveStarsWithData;
```

```
fiveStarMovies: {movieID: int,avgRating: double}
nameLookup: {movieID: int,movieTitle: chararray,releaseTime: long}

fiveStarsWithData: {fiveStarMovies::movieID: int,fiveStarMovies::avgRating: double,
        nameLookup::movieID: int,nameLookup::movieTitle: chararray,nameLookup::releaseTime: long}

(12,4.385767790262173,12,Usual Suspects, The (1995),808358400)
(22,4.151515151515151,22,Braveheart (1995),824428800)
(23,4.120879120879120 4,23,Taxi Driver (1976),824428800)
```

ORDER BY

```
oldestFiveStarMovies = ORDER fiveStarsWithData BY
      nameLookup::releaseTime;

DUMP oldestFiveStarMovies;
```

```
(493,4.15,493,Thin Man, The (1934),-1136073600)
(604,4.012345679012346,604,It Happened One Night (1934),-1136073600)
(615,4.0508474576271185,615,39 Steps, The (1935),-1104537600)
(1203,4.047619047619047,1203,Top Hat (1935),-1104537600)
```

Putting it all together

```
ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);

metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
    AS (movieID:int, movieTitle:chararray, releaseDate:chararray, videoRelease:chararray, imdbLink:chararray);

nameLookup = FOREACH metadata GENERATE movieID, movieTitle,
    ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) AS releaseTime;

ratingsByMovie = GROUP ratings BY movieID;

avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID, AVG(ratings.rating) AS avgRating;

fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;

fiveStarsWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;

oldestFiveStarMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;

DUMP oldestFiveStarMovies;
```

Let’s run it

Pig Latin: Diving Deeper
Things you can do to a relation

- LOAD STORE DUMP
  - *STORE ratings INTO 'outRatings' USING PigStorage(':');*
- FILTER DISTINCT FOREACH/GENERATE MAPREDUCE STREAM SAMPLE
- JOIN COGROUP GROUP CROSS CUBE
- ORDER RANK LIMIT
- UNION SPLIT

Diagnostics

- DESCRIBE
- EXPLAIN
- ILLUSTRATE

UDF’s

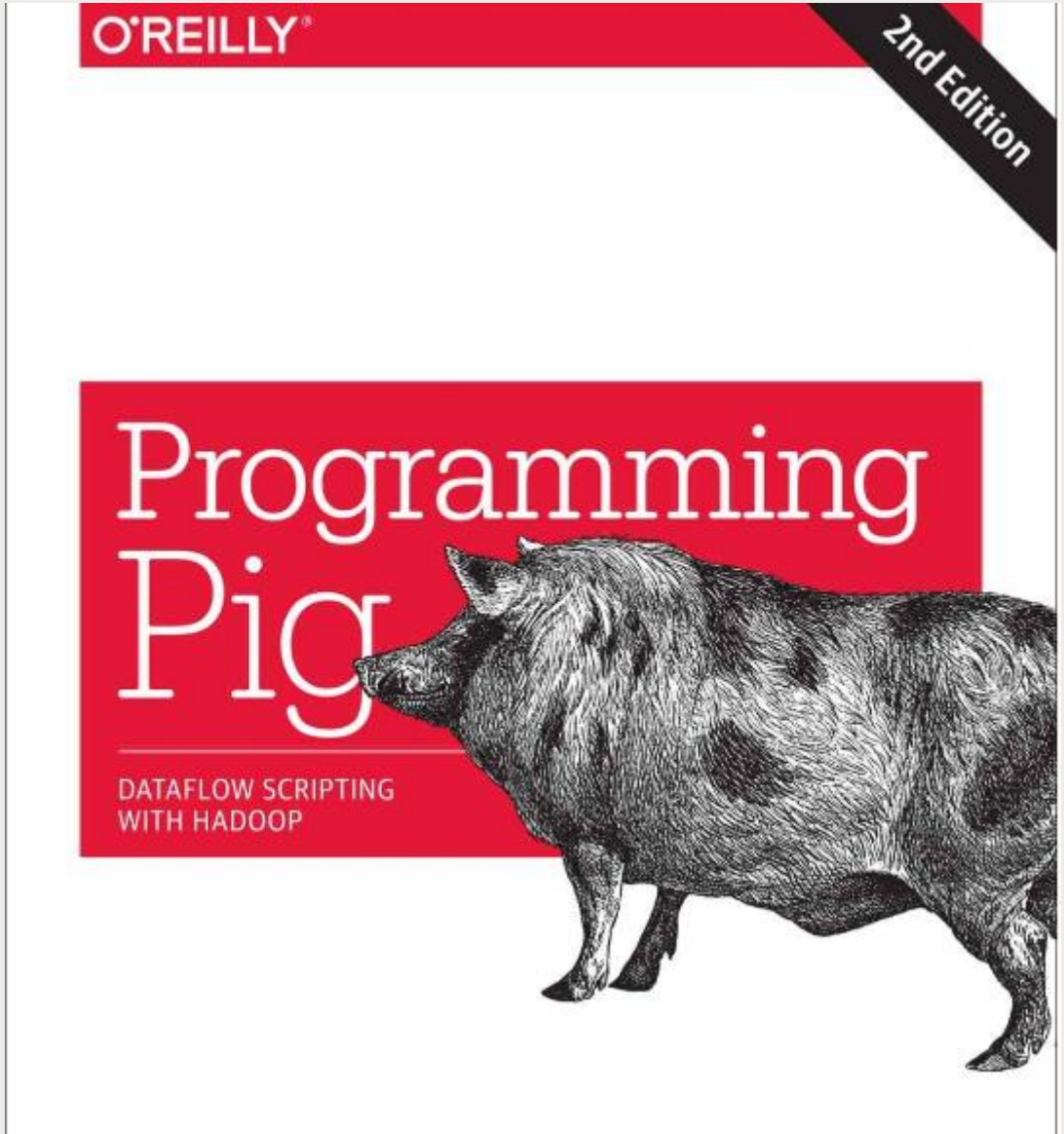
- REGISTER
- DEFINE
- IMPORT

Some other functions and loaders

- AVG CONCAT COUNT MAX MIN SIZE SUM

- PigStorage
- TextLoader
- JsonLoader
- AvroStorage
- ParquetLoader
- OrcStorage
- HBaseStorage

Learning more



PIG CHALLENGE

Find the most popular bad movies

Defining the problem

- Find all movies with an average rating less than 2.0
- Sort them by the total number of ratings

Hint

- We used everything you need in our earlier example of finding old movies with ratings greater than 4.0
- Only new thing you need is COUNT(). This lets you count up the number of items in a bag.
  - *So just like you can say AVG(ratings.rating) to get the average rating from a bag of ratings,*
  - *You can say COUNT(ratings.rating) to get the total number of ratings for a given group's bag.*