JOHNS HOPKINS
CAREY BUSINESS SCHOOL

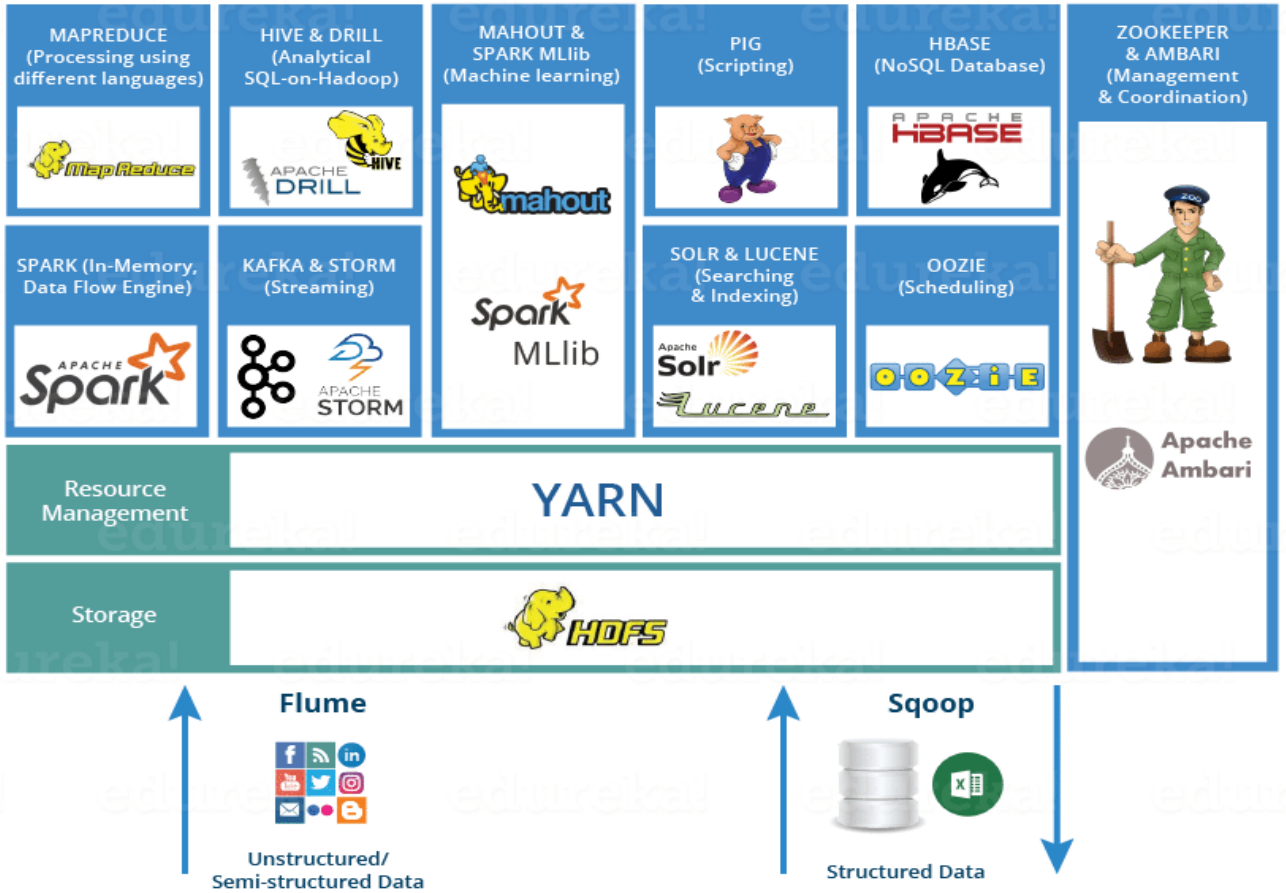
Apache Pig

BU.330.740 Large Scale Computing on the Cloud

Minghong Xu, PhD.
Associate Professor

Hadoop Overview

- An open source framework for writing and running distributed applications that process large amount of data
- Google: first to publicize MapReduce for scaled data processing
- Doug Cutting: develop the first version of Hadoop



Key Components in this Course

- Distributed File System: HDFS
- Operating System: YARN
- Original Distributed Processing Engine: MapReduce
- Improved MapReduce: Spark
- Distributed Query Language: Hive
- Distributed Scripting Language: Apache Pig

JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Pig: Distributed Scripting Language

Scripting Language and Apache Pig

- Scripting language: perform dataflow actions
  - Series of commands
  - Commonly used scripting languages: JavaScript, PHP, Python, Perl, R, …
  - Different from Java, a programming language

- Pig engine converts the queries into MapReduce jobs
- Enables people to focus more on analyzing bulk data sets and to spend less time writing Map-Reduce programs

Scripting vs Programming

- Platform specific
  - Scripting language are platform-specific, while programing languages are cross-platform (ability to execute themselves)
- Interpreted vs Complied
  - Programing languages are complied, scripting languages are mostly interpreted
  - Python can be both complied and interpreted
- Speed
  - Programming languages run faster than scripting languages
- Sometimes used interchangeably

Pigs are lazy and smart

- Optimize your codes
  - No loading data to the field until use
  - Change step sequence if more efficient
- Will not do anything until output (DUMP/STORE)is required
  - Load data is fast; only caches the command rather than execute it
- Similar to Pigs, who eat anything, the Pig programming language is designed to work upon any kind of data. That's why the name, Pig!

Load

- load **statement**
  - **Pig's default loading function is called** PigStorage
  - PigStorage **assumes text format with tab-separated columns**
    - Alternative delimiter can also be used

```
Alice      2999
Bob        3625
Carlos     2764
```

```
allsales = LOAD 'sales.csv' USING PigStorage(',') AS
(name, price);
```

- **This example loads data from the above file**

```
allsales = LOAD 'sales' AS (name, price);
```

```
allsales = LOAD 'sales.txt' USING PigStorage('|');
```

Output

- store statement: send output to disk (HDFS)
  - Output path is the name of a directory, that must not yet exist

```
STORE bigsales INTO 'myreport';
```

```
STORE bigsales INTO 'myreport' USING PigStorage(',');
```

- dump statement: send output to the screen
  - Not used on AWS in this class

Filter and Distinct

- filter: select which records will be retained

```
allsales = LOAD 'sales' AS (name, price);
bigsales = FILTER allsales BY price > 999;
STORE bigsales INTO 'myreport';
```

- distinct: removes duplicate records
  - All fields must be equal

```
unique_records = DISTINCT all_alices;
```

all_alices

firstname	lastname	country
Alice	Smith	us
Alice	Jones	us
Alice	Brown	us
Alice	Brown	us
Alice	Brown	ca

unique_records

firstname	lastname	country
Alice	Smith	us
Alice	Jones	us
Alice	Brown	us
Alice	Brown	ca

Foreach

- foreach: **apply to every record in the data pipeline**

```
twofields = FOREACH allsales GENERATE amount, trans_id;
```

allsales

salesperson	amount	trans_id
Alice	2999	107546
Bob	3625	107547
Carlos	2764	107548
Dieter	1749	107549
Étienne	2368	107550
Fredo	5637	107550

twofields

amount	trans_id
2999	107546
3625	107547
2764	107548
1749	107549
2368	107550
5637	107550

Group

- group: collect records with the same key
  - Produce nested data structures

```
grunt> byname = GROUP sales BY name;
grunt> DUMP byname;
(Bob,{(Bob,3999)})
(Alice,{(Alice,729),(Alice,27999)})
(Carol,{(Carol,32999),(Carol,4999)})

grunt> totals = FOREACH byname GENERATE
                                group, SUM(sales.price);
grunt> dump totals;
(Bob,3999)
(Alice,28728)
(Carol,37998)
```

Flatten

- flatten **operator removes a level of nesting in data**

```
grunt> byname = GROUP sales BY name;
grunt> DUMP byname;
(Bob,{(Bob,3999)})
(Alice,{(Alice,729),(Alice,27999)})
(Carol,{(Carol,32999),(Carol,4999)})

grunt> flat = FOREACH byname GENERATE group,
       FLATTEN(sales.price);
grunt> DUMP flat;
(Bob,3999)
(Alice,729)
(Alice,27999)
(Carol,32999)
(Carol,4999)
```

Join

- join: **inner/left outer/right outer/full outer**

```
daily = load 'NYSE_daily' as (exchange, symbol, date, open, high, low, close,
                              volume, adj_close);
divs  = load 'NYSE_dividends' as (exchange, symbol, date, dividends);
jnd   = join daily by symbol, divs by symbol;
dump jnd;
```

```
daily = load 'NYSE_daily' as (exchange, symbol, date, open, high, low, close,
                              volume, adj_close);
divs  = load 'NYSE_dividends' as (exchange, symbol, date, dividends);
jnd   = join daily by (symbol, date) left outer, divs by (symbol, date);
dump jnd;
```

Sample and Parallel

- sample: **get a sample of data**

```
divs = load 'NYSE_dividends';
some = sample divs 0.1;
dump some;
```

- parallel: **parallel clause can be attached to any relational operator, such as** group, order, distinct, join, limit…
  - Control only reduce-side parallelism

```
daily   = load 'NYSE_daily' as (exchange, symbol, date, open, high, low, close,
                                volume, adj_close);
bysymbl = group daily by symbol parallel 10;
average = foreach bysymbl generate group, AVG(daily.close) as avg;
sorted  = order average by avg desc parallel 2;
dump sorted;
```

Lab 3 Extension

- Movie review data from MovieLens: a tab separated list of

user id | item id | rating | timestamp

- Use item-item scheme
- Remove niche movies (to alleviate sparsity issue)
  - Having less than 30 ratings
  - Having less than 30 co-ratings
- Use correlation as similarity measure

$$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

Load Data and Pre-processing

```
-- Loading base data
movies_ratings = LOAD '$INPUT' USING PigStorage('\t') AS (user_id:int, movie_id:int, rating:int) ;
-- Starting by limiting the dataset to movies with at least 30 ratings ;
B = GROUP movies_ratings BY movie_id ;
C = FOREACH B GENERATE group AS movie_id, COUNT($1) AS count ;
D = FILTER C BY count >= 30 ;
E = FOREACH D GENERATE movie_id AS movie_ok ;
F = JOIN movies_ratings BY movie_id, E BY movie_ok ;
```

movies_ratings: load data from input location with 3 fields: user_id, movie_id and rating all as integer, separate values by tab

B: collect movie rating records with the same movie_id

C: for each movie_id, count the 2nd field in B. Note: $1 is positional notation, starting from 0, so $1 means 2nd position

D: select records in C, those count >= 30 will be retained

E: change column name in D from movie_id to movie_ok

F: inner join movies_ratings with E, retain all the fields

Create Co-ratings

```
-- Create 2 filtered datasets for self-join;
filtered = FOREACH F GENERATE user_id, movie_id, rating ;
filtered_2 = FOREACH F GENERATE user_id AS user_id_2, movie_id AS movie_id_2, rating AS rating_2 ;
-- Creating co-ratings with a self join ;
pairs = JOIN filtered BY user_id, filtered_2 BY user_id_2 ;
-- Eliminate dupes ;
J = FILTER pairs BY movie_id < movie_id_2 ;
```

filtered: select 3 fields (user_id, movie_id and rating) from F

filtered_2: also select 3 fields (user_id, movie_id and rating) from F, but name these as user_id_2, movie_id_2 and rating_2

pairs: inner join filtered with filtered_2, by the same user_id

J: select some records from pairs, for which movie_id is smaller than movie_id_2

Process Co-ratings

```
-- Corating data ;
K = FOREACH J GENERATE
                movie_id ,
                movie_id_2 ,
                rating ,
                rating_2 ,
                rating * rating AS ratingSq ,
                rating_2 * rating_2 AS rating2Sq ,
                rating * rating_2 AS dotProduct ;
L = GROUP K BY (movie_id, movie_id_2) ;
co = FOREACH L GENERATE
                group ,
                COUNT(K.movie_id) AS N ,
                SUM(K.rating) AS ratingSum ,
                SUM(K.rating_2) AS rating2Sum ,
                SUM(K.ratingSq) AS ratingSqSum ,
                SUM(K.rating2Sq) AS rating2SqSum ,
                SUM(K.dotProduct) AS dotProductSum ;
coratings = FILTER co BY N >= 30 ;
```

K: for each pair of co-rating, calculate rating^2, rating_2^2, and rating*rating_2

$$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

L: collect co-ratings with the same (movie_id, movie_id_2)

co: for each (movie_id, movie_id_2), calculate count of co-rating pairs for movie_id, sum of rating, sum of rating_2, sum of rating^2, sum of rating_2^2, and sum of rating*rating_2

coratings: select co-ratings which has at least 30 shared users

Calculate Recommendations

```
recommendations = FOREACH coratings GENERATE
                group.movie_id ,
                group.movie_id_2 ,
                (double)(N * dotProductSum - ratingSum * rating2Sum) / ( SQRT((double)(N *
ratingSqSum - ratingSum * ratingSum)) * SQRT((double)(N * rating2SqSum - rating2Sum *
rating2Sum)) ) AS correlation ;

Store recommendations into '$OUTPUT';
```

recommendations: calculate correlation between movies, using equation $\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$