PIG

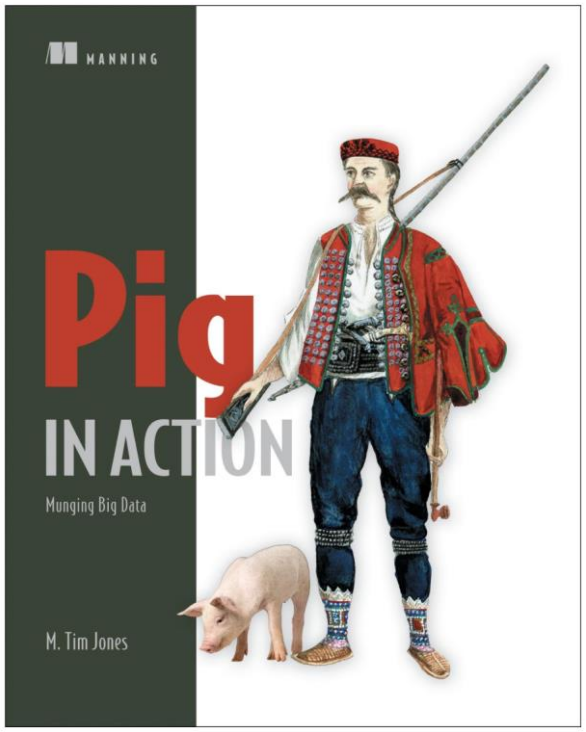
DS8003 – MGT OF BIG DATA AND TOOLS

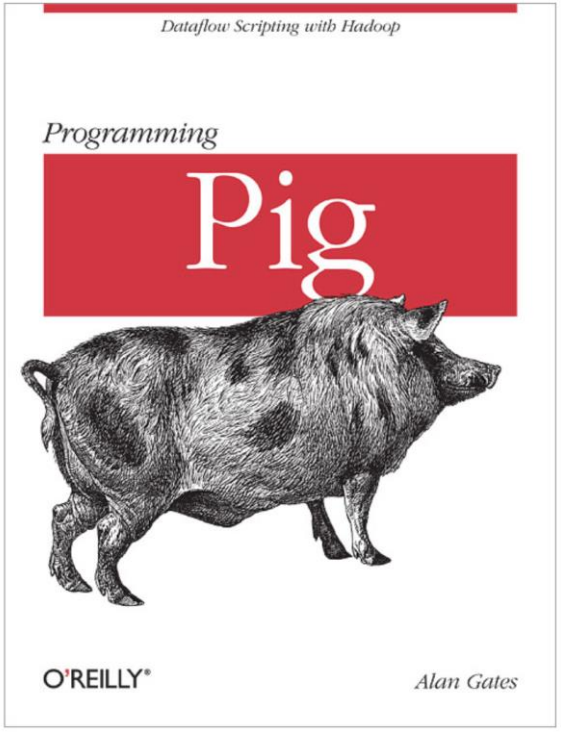
Ryerson University

Instructor: Kanchana Padmanabhan

Apache Pig

Learning Pig

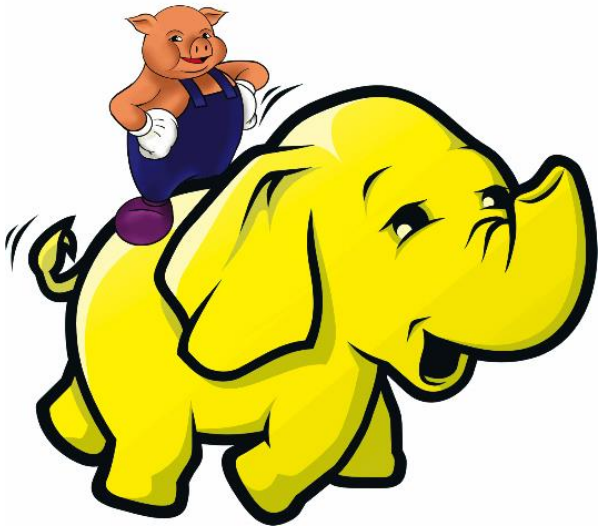




Your best friend? Apache Pig Online Manual → http://pig.apache.org/docs/r0.14.0/

Apache Pig

- Pig Latin, a high level data processing language.
- An engine that executes Pig Latin locally or on a Hadoop cluster



Pig Latin Example – Word Count

This is a relation, NOT a variable

loads a file into a relation

file schema

```
a = LOAD '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);
```

projection operation

explode function

```
b = FOREACH a GENERATE FLATTEN(TOKENIZE(tweet)) AS token;
```

group

tokenization function

```
c = GROUP b BY token;
```

count function

```
d = FOREACH c GENERATE group AS token, COUNT_STAR(b) AS cnt;
```

order operation

```
e = ORDER d BY cnt DESC;
```

limit operation

```
f = LIMIT e 20;
```

```
DUMP f;
```

pig doesn't execute until it sees keyword such as dump or store

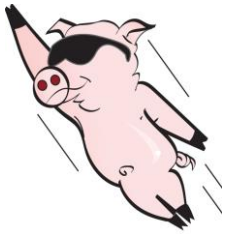
http://hadooptutorial.info/pig-functions-examples/

Why Pig?

- Faster development
  - Fewer lines of code
  - Don't re-invent the wheel
  - No M/R programming
  - Joins in M/R is painful
  - Chaining together M/R jobs is tedious
- Flexible
  - Metadata is optional
  - Extensible (UDFs → Piggybank, DataFu, etc.)
  - Procedural programming

Pig Philosophy

- Pigs eats anything
  - Pig can operate on data whether it has metadata or not. It can operate on data that is relational, nested, or unstructured. And it can easily be extended to operate on data beyond files, including key/value stores, databases, etc.
- Pigs live anywhere
  - Pig is intended to be a language for parallel data processing. It is not tied to one particular parallel framework (e.g., Pig on Spark, Pig on Storm)
- Pigs are domestic animals
  - Designed to be easily controlled and modified by users via User-Defined-Functions (UDF) and Stream command, etc.



Apache Pig Philosophy: http://pig.apache.org/philosophy.html

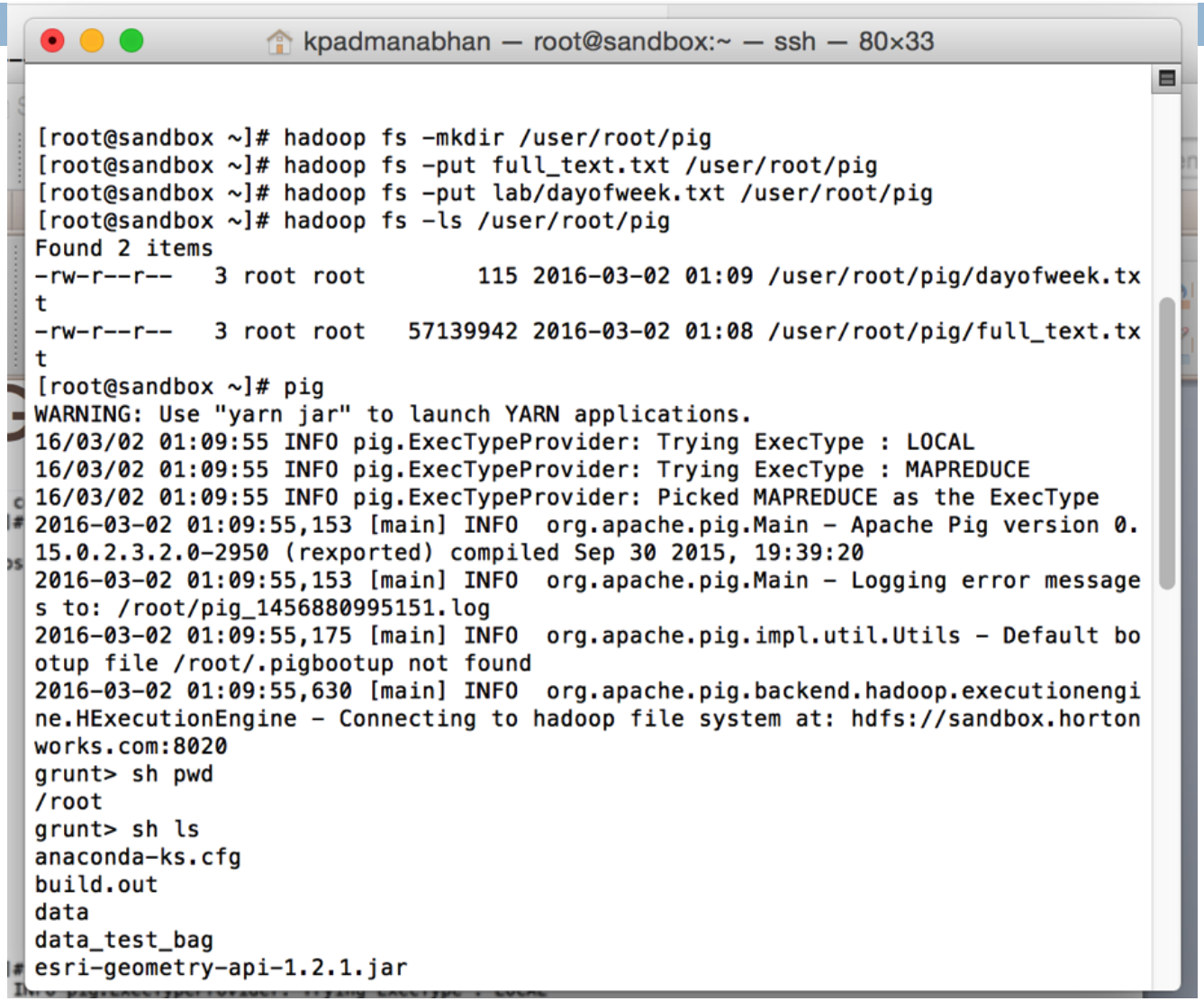
Pig Grunt/Shell

Linux	Pig Grunt Shell	FsShell (working with HDFS from pig grunt)

$ pig -x local
$ pig
$ pig -e script.pig
$ pig –param YEAR=2015 script.pig
	grunt> sh pwd
	grunt> sh ls -alF
	/home/lab/grunt> sh cat test.pig
		grunt> fs -ls /user
		grunt> fs -ls /user/root/pig
		grunt> fs -put /root/lab/full_text.txt /user/root/pig/full_text_1.txt
		grunt> copyFromLocal
		grunt> copyToLocal
		grunt> rmf filename
		grunt> kill jobid
		grunt> exec test1.pig
		grunt> run test1.pig
		grunt> describe fieldA;

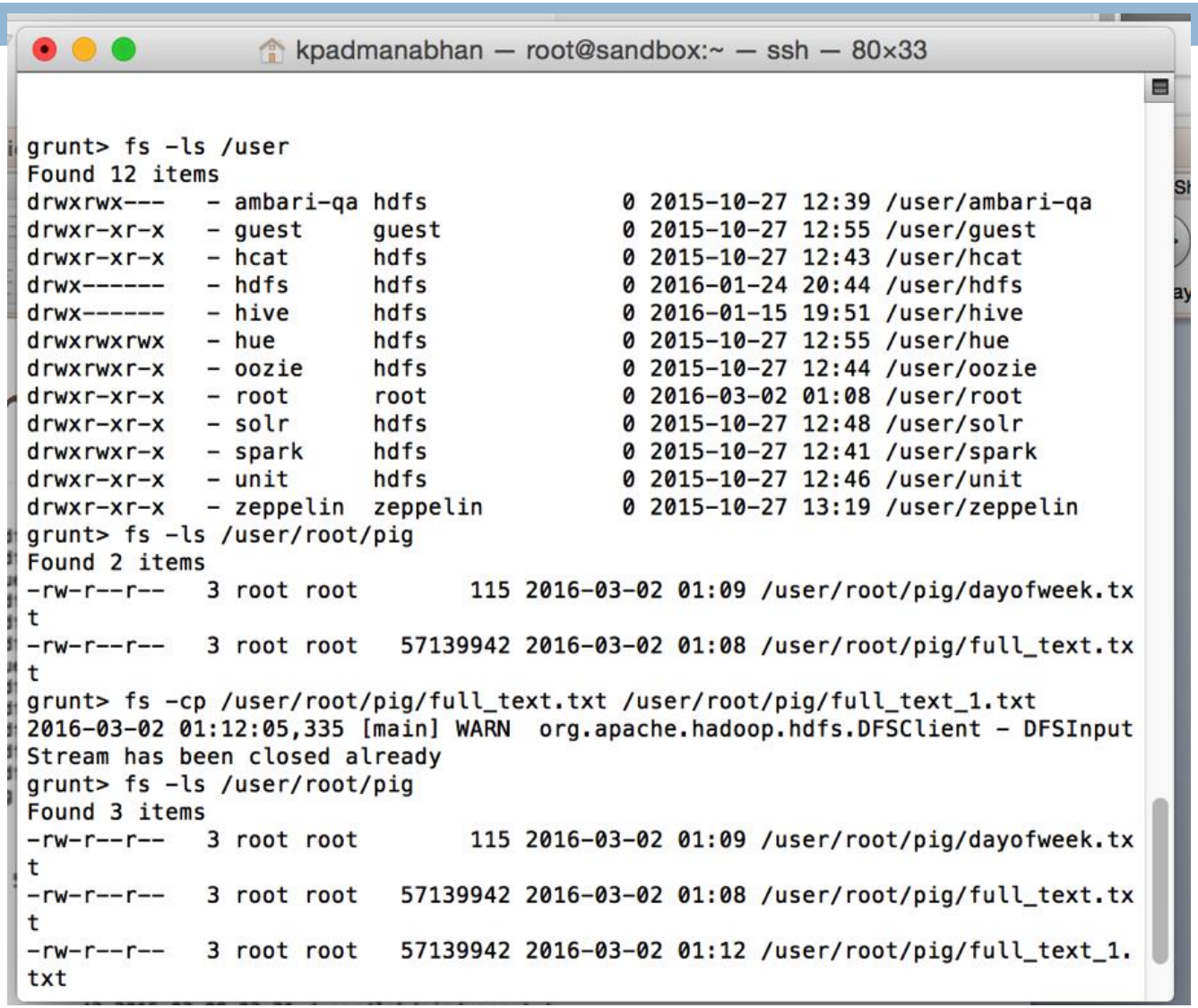
DEMO

Data Preparation



DEMO

Pig Grunt Shell Demo



Pig Operations

- load/store
- Date/Time/String
- foreach
- filter
- group
- limit
- order by
- sample
- join

Relations

- Pig's fundamental building blocks
- Similar to a **table**
- Don't confuse it with variables in other languages

```
a = LOAD '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);
```

ID	DateTime	Latitude	Longitude	Tweet
USER_8d0e8566	2010-03-02T23:00:44	30.387524	-91.109663	Pre-workout prep has begun.
USER_8d0e8566	2010-04-02T23:04:20	30.387524	-91.109663	I really don't like that a college FB player's ON-FIELD production can be negated by a single workout. So proof's NOT in the pudding?
USER_87b48222	2010-03-02T23:23:29	37.530819	-77.475577	@USER_9bb099c2 15 pages??? fuck u mean!!?? damn.
USER_87b48222	2010-07-02T23:43:57	37.530819	-77.475577	@USER_e97d1292 lol do u know that song?

DEMO

Load Data


```
a = LOAD '/user/root/pig/full_text.txt' USING
PigStorage('\t') AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);
b = foreach a generate id, ts, lat, lon;
c = limit b 5;
DUMP c;
```

```
(USER_79321756,2010-03-03T04:15:26,47.528137,-122.197914)
(USER_79321756,2010-03-03T04:55:32,47.528137,-122.197914)
(USER_79321756,2010-03-03T05:13:34,47.528137,-122.197914)
(USER_79321756,2010-03-03T05:28:02,47.528137,-122.197914)
(USER_79321756,2010-03-03T05:56:13,47.528137,-122.197914)
```

DEMO

Store Data


```
a = LOAD '/user/root/pig/full_text.txt' USING PigStorage('\t') AS
(id:chararray, ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);

b = foreach a generate id, ts, lat, lon;

c = limit b 5;

STORE c INTO '/user/root/pig/full_text_proj' USING PigStorage('#') ;
```

```
USER_79321756#2010-03-03T04:15:26#47.528137#-122.197914
USER_79321756#2010-03-03T04:55:32#47.528137#-122.197914
USER_79321756#2010-03-03T05:13:34#47.528137#-122.197914
USER_79321756#2010-03-03T05:28:02#47.528137#-122.197914
USER_79321756#2010-03-03T05:56:13#47.528137#-122.197914
```

DEMO

Foreach (Projection)

- Apply transformations on every row in a relation
- Projections can be chained
- Creates a new relation
- Analogous to “select” statement in Hive

```
a = LOAD '/user/root/pig/full_text.txt' USING PigStorage('\t') AS
(id:chararray, ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);
b = FOREACH a GENERATE id, To_Date(ts), (lat, lon) as location;
c = limit b 5;
STORE c INTO '/user/root/pig/full_text_proj' USING PigStorage('#') ;
```

Filtering Data

- An evaluation is done for each row in a relation
  - The row is tossed if it does not meet certain criteria
- Can be used with boolean logic, regular expression etc.

```
b = FILTER a by id=='1234567'
```

For more on pig comparison operators, check out
http://pig.apache.org/docs/r0.14.0/basic.html#comparison

Filtering Data

DEMO

find retweets in NYC on 12th with length smaller than 50 characters


```
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray,
lat:float, lon:float, tweet:chararray);
b = FILTER a by lat > 40.4774 and lat < 40.9176 and lon > -74.2589 and lon < -
73.7004 and
        SIZE(tweet)<50 and
        GetHour(ToDate(ts))==12 and
        tweet matches 'RT.*';
c = foreach b generate id, ts, lat, lon, tweet;
d = limit c 10;
dump d;
```

```
(USER_a8cecd53,2010-03-04T12:44:36,40.812267,-73.95978,RT @USER_a55ef2d4: on da dam bus goin to skool)
(USER_f8eb395d,2010-03-03T12:37:19,40.677395,-73.87458,RT @USER_ca1182eb: Hate has no color or age!)
(USER_827611e3,2010-03-04T12:40:03,40.75,-73.997,RT @USER_a2a7b7e5: Watching Spongebob !)
(USER_18c466a9,2010-03-06T12:54:41,40.899567,-73.85717,RT @USER_86e28cc0 hella bored - Tweet people)
(USER_838d6d62,2010-03-05T12:37:48,40.672306,-73.92697,RT @USER_cb2f3989: @USER_838d6d62 smh/im up)
(USER_838d6d62,2010-03-05T12:52:00,40.672306,-73.92697,RT @USER_231d82a1: @USER_838d6d62 gud mornin/:))
(USER_0b58828b,2010-03-04T12:19:51,40.83705,-73.86019,RT @USER_32550f0c: Good Morning | goodmorning)
(USER_c6710d1e,2010-03-03T12:45:57,40.664516,-73.945206,RT @USER_e517c738: yeah im about to let him go!)
```

Comparison Operator


```
data = load '/user/root/pig/full_text.txt' AS (id:chararray,
ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);
filtr = FILTER data BY tweet MATCHES '.*\\#\\p{{Alpha}}.*?';
limt = limit filtr 20;
dump limt;
```

Group (aggregation)

- The group statement collects together records with the same key into a bag
- Grouping does not change the data
  - Reorganizes it based on the given key
  - Can group on multiple keys
- First column is always called **group**
  - A compound group key will be a Tuple ("group") whose elements are the keys
- Second column is a Bag
  - Name is the grouped relation
  - Contains every row associated with key

DEMO

Group (aggregation)


```
-- Calculate # of tweets per user

a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);
b = GROUP a BY id;
c = FOREACH b GENERATE group AS id, COUNT(a) as cnt;
d = order c by cnt desc;
e = limit d 5;
dump e;

-- visualize group
illustrate b;
```

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
| a     | id:chararray      | ts:chararray         | location:chararray        | lat:float      | lon:float      | tweet:chararray                                    |
--------------------------------------------------------------------------------------------------------------------------------------------------------------
|       | USER_c296a14f     | 2010-03-03T13:41:06 | ?T: 42.133812,-87.887853 | 42.133812      | -87.887856     | @USER_b8034b39 eh? You dnt NEVER be on. How u doin love?   |
|       | USER_c296a14f     | 2010-03-03T06:05:51 | ?T: 42.133812,-87.887853 | 42.133812      | -87.887856     | @USER_7b31238d lmao nawww not a hater -- a regulator. [smh] |
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| b     | group:chararray      | a:bag{:tuple(id:chararray,ts:chararray,location:chararray,lat:float,lon:float,tweet:chararray)}                                               |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|       | USER_c296a14f        | {(USER_c296a14f, ..., @USER_b8034b39 eh? You dnt NEVER be on. How u doin love?), (USER_c296a14f, ..., @USER_7b31238d lmao nawww not a hater -- a regulator. [smh])} |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

DEMO

Group (aggregation)

-- Count total number of records

```
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);
b = GROUP a ALL;
c = FOREACH b GENERATE COUNT_STAR(a);
dump c;

-- visualize relation b
illustrate b;
```

```
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
| a     | id:chararray       | ts:chararray        | location:chararray        | lat:float       | lon:float       | tweet:chararray                          |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
|       | USER_06f07cc0      | 2010-03-05T23:29:11 | ?T: 40.805065,-73.563726  | 40.805065       | -73.56373       | My reflection has a hard time being me. |
|       | USER_6ccd47dd      | 2010-03-03T19:37:11 | ?T: 44.968375,-124.012122 | 44.968376       | -124.01212      | @USER_3903ee32 dopppeee!!!               |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
| b     | group:chararray       | a:bag{:tuple(id:chararray,ts:chararray,location:chararray,lat:float,lon:float,tweet:chararray)}
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
|       | all                   | {(USER_06f07cc0, ..., My reflection has a hard time being me.), (USER_6ccd47dd, ..., @USER_3903ee32 dopppeee!!!)}
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Sampling

-- approach 1

```
grunt> data = load 'file';
grunt> sample1= SAMPLE data 0.1;
```

-- approach 2

```
grunt> data = load 'file';
grunt> sample2 = FILTER A by RANDOM() <= 0.1;
```

-- approach 3

```
grunt> data = load 'file';
grunt> grpd = group data all;
grunt> grpd_cnt = foreach grpd generate COUNT(data) as num_rows;
grunt> sample3 = SAMPLE grpd_cnt 10000/grpd_cnt .num_rows;
```

Join

find tweets on weekends


```
-- prepare lookup table 'dayofweek'
fs -put /root/lab/dayofweek.txt /user/root/pig/

-- Find Weekend Tweets (inner join)
a = load '/user/root/pig/full_text.txt' using PigStorage('\t') AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);
a1 = foreach a generate id, ts, SUBSTRING(ts,0,10) as date;

b = load '/user/root/pig/dayofweek.txt' using PigStorage('\t') as (date:chararray,
dow:chararray);
b1 = filter b by dow=='Saturday' or dow=='Sunday';

c = JOIN a1 BY date, b1 BY date;
d = foreach c generate a1::id .. a1::date, b1::dow as dow;
e = limit d 5;
dump e;
```

```
(USER_606adf97,2010-03-06T08:16:55,2010-03-06,Saturday)
(USER_8c704efa,2010-03-06T17:51:27,2010-03-06,Saturday)
(USER_8c704efa,2010-03-06T18:49:45,2010-03-06,Saturday)
(USER_8c704efa,2010-03-06T19:03:48,2010-03-06,Saturday)
(USER_8c704efa,2010-03-06T19:19:05,2010-03-06,Saturday)
```

Advanced Joins

- replicated join
- skewed join
- semi-join (co-group)
- non-equi join (cross)

Replicated Join

- Joins small data to large data
- Often used when a lookup file is in a smaller file
  - Small enough to fit in memory of your computer node
- Using **distributed cache**
- Replicated join applies map-only join
  - No reduce phase – Yeaaah!!
- Supports only inner and left outer join
- Can be used with more than two tables.
- In this case, all but the first (left-most) table are read into memory
  - The lookup file should always be on the right side

Replicated JOIN Demo

Finding Weekend Tweets


```
a = load '/user/root/pig/full_text.txt' using PigStorage('\t') AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float, tweet:chararray);
a1 = foreach a generate id, ts, SUBSTRING(ts,0,10) as date;

b = load '/user/root/pig/dayofweek.txt' using PigStorage('\t') as (date:chararray, dow:chararray);
b1 = filter b by dow=='Saturday' or dow=='Sunday';

c = join a1 by date, b1 by date using 'replicated';
d = foreach c generate a1::id .. a1::date, b1::dow as dow;
e = limit d 5;
dump e;
```

Skewed Join

c = join a by key, b by key using 'skewed';

- Skewed join is used when the data has significant skew in the number of records per key. This results in one or two reducers that will take much longer than the rest → Straggler problem
- Adds 5% overhead to the performance due to the sampling approach at the beginning
- Skew join works by first sampling one input for the join. In that input it identifies any keys that have so many records that skew join estimates it will not be able to fit them all into memory.
- Those keys identified as too large are treated differently. Based on how many records were seen for a given key, those records are split across the appropriate number of reducers. The number of reducers is chosen based on Pig's estimate of how wide the data must be split such that each reducer can fit its split into memory. For the input to the join that is not split, those keys that were split are then replicated to each reducer that contains that key.

Non-Equi Join (CROSS) – Data Prep

- [root@sandbox ~]# echo -e "nfriends,2\nndays,100\nnvisits,13" > /root/lab/params.txt
- [root@sandbox ~]# echo -e "Amy,George\nGeorge,Fred\nFred,Anne\nGeorge,Joe\nGeorge,Harry" > /root/lab/friend.txt
- [root@sandbox ~]# hadoop fs -put params.txt /user/root/pig/
- [root@sandbox ~]# hadoop fs -put friend.txt /user/root/pig/

Non-Equi Join (CROSS)

find user with more than 2 friends

p_name	value
nfriends	2
ndays	100
nvists	13

name	friend
Amy	George
George	Fred
Fred	Anne
George	Joe
George	Harry

```
grunt> params = load '/user/root/pig/params.txt' using PigStorage(',') as
(p_name:chararray, value:int);
grunt> friend = load '/user/root/pig/friend.txt' using PigStorage(',') as
(name:chararray, friend:chararray);
grunt> friend_grp = group friend by name;
grunt> friend_cnt = foreach friend_grp generate group as name, COUNT(friend.friend)
as cnt;
```

name	friend
Amy	1
George	3
Fred	1

2

```
grunt> friend_param = filter params by p_name=='nfriends';
grunt> friend_param_p = foreach friend_param generate value;
```

name	friend	fr_param_p
Amy	1	2
George	3	2
Fred	1	2

```
grunt> friend_cross = CROSS friend_cnt, friend_param_p;

grunt> friend_cross_1 = filter friend_cross by friend_cnt::cnt >= friend_param_p::value;
grunt> dump friend_cross_1;
```

George	3	2

COGROUP

- COGROUP is a generalization of GROUP that works with more relations
- You can COGROUP up to but no more than 127 relations at a time
- The result is a record with a key and one bag for each input. Each bag contains all records from that input that have the given value for the key
- Another way to think of COGROUP is as the first half of a join. The keys are collected together, but the cross product is not done
- COGROUP is useful when you want to do join-like things but not a full join

COGROUP – DATA PREP

- [root@sandbox ~]# echo -e "u1,14,M,US\nu2,32,F,UK\nu3,22,M,US" > /root/lab/user.txt
- [root@sandbox ~]# hadoop fs -put /root/lab/user.txt /user/root/pig/user.txt
- [root@sandbox ~]# echo -e "u1,US\nu1,UK\nu1,CA\nu2,US" > /root/lab/session.txt
- [root@sandbox ~]# hadoop fs -put /root/lab/session.txt /user/root/pig/session.txt

COGROUP Demo

uid	age	gender	region
u1	14	M	US
u2	32	F	UK
u3	22	M	US

uid	region
u1	US
u1	UK
u1	CA
u2	US


```
grunt> user = load '/user/root/pig/user.txt' using PigStorage(',') as
(uid:chararray, age:int, gender:chararray, region:chararray);
grunt> session = load '/user/root/pig/session.txt' using PigStorage(',') as
(uid:chararray, region:chararray);

grunt> cogrp = COGROUP user BY uid, session BY uid;
grunt> describe cogrp;
cogrp: {group: chararray,user: {(uid: chararray,age: int,gender: chararray,region:
chararray)},session: {(uid: chararray,region: chararray)}}

grunt> dump cogrp;
(u1,{(u1,14, M, US)}, {(u1, CA),(u1, UK),(u1, US)})
(u2,{(u2,32, F, UK)}, {(u2, US)})
(u3,{(u3,22, M, US)}, {})

grunt> cogrp_nest = foreach cogrp {
    crossed = cross user, session;
    generate crossed;
}
grunt> dump cogrp_nest;
({(u1,14, M, US, u1, CA), (u1,14, M, US, u1, UK), (u1,14, M, US, u1, US)})
({(u2,32, F, UK,u2, US)})
```

Set Intersection – Data Prep

- [root@sandbox ~]# echo -e "John,3\nHarry,4\nGeorge,2" > /root/lab/s1.txt
- [root@sandbox ~]# echo -e "John,2\nJohn,3\nGeorge,0\nSue,1" > /root/lab/s2.txt
- [root@sandbox ~]# hadoop fs -put s1.txt /user/root/pig/
- [root@sandbox ~]# hadoop fs -put s2.txt /user/root/pig/

Set Intersection

with COGROUP

name	hits
John	3
Harry	4
George	2

name	errors
John	2
John	3
George	0
Sue	1


```
grunt> s1 = load '/user/root/pig/s1' using PigStorage(',') as (name:chararray, hits:int);
grunt> s2 = load '/user/root/pig/s2' using PigStorage(',')  as (name:chararray,
errors:int);

grunt> grps = COGROUP s1 BY name, s2 BY name;

(John, {(John, 3)}, {(John, 2), (John, 3)})
(Harry, {(Harry, 4)}
(George, {(George, 2), (George, 2)})
(Sue, { }, {(Sue, 1)})

-- Note:
-- Something is in the intersection of s1 and s2 if there are no {}'s in the cogroup.

grunt> grp2 = FILTER grps by NOT(IsEmpty(s1)) AND NOT(IsEmpty(s2));
grunt> intersection = FOREACH grp2 GENERATE group as grp, s1, s2 ;
grunt> dump intersection;

(John,{(John,3)},{(John,3),(John,2)})
(George,{(George,2)},{(George,0)})
```

Set Difference

with COGROUP

name	hits
John	3
Harry	4
George	2

name	errors
John	2
John	3
George	0
Sue	1


```
grunt> s1 = load '/user/root/pig/s1' using PigStorage(',') as (name:chararray, hits:int);
grunt> s2 = load '/user/root/pig/s2' using PigStorage(',')  as (name:chararray,
errors:int);

grunt> grps = COGROUP s1 BY name, s2 BY name;

(John, {(John, 3)}, {(John, 2), (John, 3)})
(Harry, {(Harry, 4)}
(George, {(George, 2), (George, 2)})
(Sue, { }, {(Sue, 1)})

-- Note:
-- Something is in the difference between s1 and s2 if there are non-empty second sets.

grunt> grps2 = FILTER grps by IsEmpty(s2);
grunt> set_diff = FOREACH grps2 GENERATE group as grp, s1, s2 ;
grunt> dump set_diff;

(Harry,{(Harry,4)},{})
```

Pig Data Types

Type	Description	Example
int, long, float, duble, chararray, bytearray, boolean, datetime	• primitive data types in pig	
tuple	• Fixed length, **ordered** set of fields, like a row with multiple columns in SQL • Allows random access • Must fit in memory	(toronto, 3) ('bob', 53, 'toronto', 'male')
bag	• An **unordered** collection of tuples • Can have tuples with differing numbers of fields • Can spill to disk (doesn't have to fit in memory	{(toronto, 3),(chicago, 5)} {('bob',53,'toronto','male'), ('sally', 23), 'george', 'montreal')}
map	• A set of key/value pairs • Key/values in a relation must be unique • Key must be chararray, data element can be any type	[city#toronto] [name#bob,age#53,city#toronto,gender#male]

Tuple

- Tuple

twitter:tuple(id:chararray, lat:double, lon:double, tweet:chararray)

twitter.id → USER_12345678

twitter.$3 → #Hadoop, big data is the new oil

twitter.($1,$2) → (40.48956, -156.22234)

twitter.(lat,lon) → (40.48956, -156.22234)

- grunt> a = load '/user/root/pig/full_text.txt' using PigStorage('\t') AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float, tweet:chararray);
- grunt> b = Foreach a generate TOTUPLE(id,ts) as t1, TOTUPLE(lat,lon) as t2;
- grunt> c = limit b 5;
- grunt> dump c;

Complex Type

- Type Construction Operators/Functions
  - Tuple construction (a, b) ➔ Same as TOTUPLE()
  - Bag construction {a, b} ➔ Same as TOBAG()
  - Map construction [a, b] ➔ Same as TOMAP()

Complex Type – Data Prep

- [root@sandbox ~] echo -e "(3,8,9) (4,5,6)\n(1,4,7) (3,7,5)\n(2,5,8) (9,5,8)" > data
- [root@sandbox ~] hadoop fs -put data /user/root/pig/data
- [root@sandbox ~]# echo -e "user1\ta\tb\tc\nuser2\ta\tb\nuser3\ta" > data_test_bag
- [root@sandbox ~]# cat data_test_bag
- [root@sandbox ~]# hadoop fs -put data_test_bag /user/root/pig/data_test_bag
- [root@sandbox ~]# echo -e "joe smith\t20\t3.5\namychen\t22\t3.2\nleo allen\t18\t2.1" > student
- [root@sandbox ~]# hadoop fs -put student /user/root/pig/student

Bag

id	field1	field2	field3
user1	a	b	c
user2	a	b	
user3	a		

```
grunt> data = load '/user/root/pig/data_test_bag' using PigStorage('\t') as
(id:chararray, f1:chararray, f2:chararray, f3:chararray);
grunt> grpd = group data ALL;

grunt> describe grpd;
grpd: {group: chararray,data: {(id: chararray,f1: chararray,f2: chararray,f3: chararray)}}

grunt> dump grpd;
(all,{(user3, a, , ), (user2, a, b, ), (user1, a, b, c)})

cnt_id = foreach grpd generate COUNT(data.id) as cnt;      ← 3
cnt_f1 = foreach grpd generate COUNT(data.$1) as cnt;      ← 3
cnt_f2 = foreach grpd generate COUNT(data.f2) as cnt;      ← 2
cnt_f3 = foreach grpd generate COUNT(data.$3) as cnt;      ← 1
```

Bag Construction

A = load '/user/root/pig/student' as (name:chararray, age:int, gpa:float);

B = foreach A generate **{(name, age)}**, **{name, age}**;

store B into '/user/root/pig/results';

Input (students):

joe smith 20 3.5

amy chen 22 3.2

leo allen 18 2.1

Output (results):

{(joe smith,20)} {(joe smith), (20)}

{(amy chen,22)} {(amy chen), (22)}

{(leo allen,18)} {(leo allen), (18)}

Map Construction

A = load '/user/root/pig/student' as (name:chararray, age:int, gpa:float);
B = foreach A generate **[name, gpa]**;
store B into '/user/root/pig/results';

Input (students):
joe smith 20 3.5
amy chen 22 3.2
leo allen 18 2.1

Output (results):
[joe smith#3.5]
[amy chen#3.2]
[leo allen#2.1]

Flatten (explode/transpose)

- **flatten** is the inverse of **group**
- flatten is pig's explode function, similar to Excel's transpose function, but is more flexible
- flatten turns tuples into columns
- flatten turns bags into rows
- flatten reference (naming)
  - After flatten, if there're ambiguous field names, you need to use the disambiguate operator **::**
- COGROUP + FLATTEN = JOIN

Flatten Tuples

DEMO

-- Calculate # of tweets per user per day

```
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float, tweet:chararray);
b = foreach a generate id, SUBSTRING(ts, 0, 10) as date, lat, lon, tweet;
c = GROUP b BY (id, date);
d = FOREACH c GENERATE FLATTEN(group) AS (id,date) , COUNT(b) as cnt;
e = order d by cnt desc;
f = limit e 5;
dump f;
```

```
(USER_18c466a9,2010-03-07,89)
(USER_943f9c88,2010-03-04,89)
(USER_b2bb70e4,2010-03-03,85)
(USER_943f9c88,2010-03-03,83)
(USER_9e14b9d7,2010-03-05,81)
```

```
-- visualize group
illustrate c;
```

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| a      | id:chararray      | ts:chararray         | location:chararray        | lat:float      | lon:float      | tweet:chararray
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|        | USER_583b948c     | 2010-03-04T23:27:10  | ?T: 33.551659,-84.563961  | 33.55166       | -84.56396      | @USER_11f14a3d what movie u looking at
|        | USER_583b948c     | 2010-03-04T23:44:35  | ?T: 33.551659,-84.563961  | 33.55166       | -84.56396      | @USER_11f14a3d U didnt look at it cause u were on twitte
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| b      | id:chararray      | date:chararray      | lat:float      | lon:float      | tweet:chararray                                                                 |
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|        | USER_583b948c     | 2010-03-04          | 33.55166       | -84.56396      | @USER_11f14a3d what movie u looking at                                          |
|        | USER_583b948c     | 2010-03-04          | 33.55166       | -84.56396      | @USER_11f14a3d U didnt look at it cause u were on twitter smh!!! Twitter nympho |
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------
| c      | group:tuple(id:chararray,date:chararray)              | b:bag{:tuple(id:chararray,date:chararray,lat:float,lon:float,tweet:chararray)}
                             |
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------
|        | (USER_583b948c, 2010-03-04)                           | {(USER_583b948c, ..., @USER_11f14a3d what movie u looking at), (USER_583b948c, ..., @USER_11f14a3d U
tter smh!!! Twitter nympho)} |
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------
```

Flatten Tuples – cont'd

DEMO


```
-- Calculate # of tweets per user per day

a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float,
lon:float, tweet:chararray);
b = foreach a generate id, SUBSTRING(ts, 0, 10) as date, lat, lon, tweet;
c = GROUP b BY (id, date);
d = FOREACH c GENERATE FLATTEN(group) AS (id,date) , COUNT(b) as cnt;
e = order d by cnt desc;
f = limit e 5;
dump f;

-- visualize group
illustrate d;
```

```
----------------------------------------------------------------------------------------------------------------------------------------------------------
| a     | id:chararray     | ts:chararray        | location:chararray        | lat:float      | lon:float      | tweet:chararray
----------------------------------------------------------------------------------------------------------------------------------------------------------
|       | USER_2dfabc44    | 2010-03-07T02:28:13 | ?T: 33.92368,-84.304425 | 33.92368       | -84.30443      | @USER_cc540e1c another non invite.... rotten.. j
|       | USER_2dfabc44    | 2010-03-07T15:51:14 | ?T: 33.92368,-84.304425 | 33.92368       | -84.30443      | @USER_a8349405 good lookin on the link
----------------------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------
| b     | id:chararray     | date:chararray     | lat:float      | lon:float      | tweet:chararray                                            |
------------------------------------------------------------------------------------------------------------------------------------
|       | USER_2dfabc44    | 2010-03-07         | 33.92368       | -84.30443      | @USER_cc540e1c another non invite.... rotten.. just rotten |
|       | USER_2dfabc44    | 2010-03-07         | 33.92368       | -84.30443      | @USER_a8349405 good lookin on the link                     |
------------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------------------------------
| c     | group:tuple(id:chararray,date:chararray)             | b:bag{:tuple(id:chararray,date:chararray,lat:float,lon:float,tweet:chararray)}
----------------------------------------------------------------------------------------------------------------------------------------------------------
|       | (USER_2dfabc44, 2010-03-07)                          | {}
|       | (USER_2dfabc44, 2010-03-07)                          | {}
----------------------------------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------
| d     | id:chararray     | date:chararray     | cnt:long      |
-----------------------------------------------------------------
|       | USER_2dfabc44    | 2010-03-07         | 2             |
-----------------------------------------------------------------
```

Flatten Bags


```
-- Flatten Bags Example
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float,
lon:float, tweet:chararray);
b = foreach a generate id, SUBSTRING(ts, 0, 10) as date, lat, lon, tweet;
c = GROUP b BY (id, date);
d = FOREACH c GENERATE FLATTEN(b) AS (id, date, lat, lon, tweet);
e = order d by id, date;
f = limit e 50;
dump f;

-- visualize group
illustrate d;
```

```
| a      | id:chararray       | ts:chararray          | location:chararray          | lat:float        | lon:float        | tweet:chararray
| USER_d54c0e1e      | 2010-03-06T14:41:08 | ?T: 37.187699,-77.344635 | 37.1877          | -77.344635       | Morning every1
| USER_d54c0e1e      | 2010-03-06T05:47:09 | ?T: 37.26009,-77.394015  | 37.26009         | -77.39401        | Aawwww bak n da college life w/ my sistas @ da Pretty Fr

| b      | id:chararray       | date:chararray        | lat:float        | lon:float        | tweet:chararray                                                    |
|        | USER_d54c0e1e      | 2010-03-06            | 37.1877          | -77.344635       | Morning every1                                                     |
|        | USER_d54c0e1e      | 2010-03-06            | 37.26009         | -77.39401        | Aawwww bak n da college life w/ my sistas @ da Pretty Fresh house party smh lol |

| c      | group:tuple(id:chararray,date:chararray)                | b:bag{:tuple(id:chararray,date:chararray,lat:float,lon:float,tweet:chararray)}
|
|        | (USER_d54c0e1e, 2010-03-06)                             | {(USER_d54c0e1e, ..., Morning every1), (USER_d54c0e1e, ..., Aawwww bak n da college life w/ my sistas
h lol)} |

| d      | id:chararray       | date:chararray        | lat:float        | lon:float        | tweet:chararray                                                    |
|        | USER_d54c0e1e      | 2010-03-06            | 37.1877          | -77.344635       | Morning every1                                                     |
|        | USER_d54c0e1e      | 2010-03-06            | 37.26009         | -77.39401        | Aawwww bak n da college life w/ my sistas @ da Pretty Fresh house party smh lol |
```

Foreach with multiple commands


```
foreach j_2 {
sort = order j_1 by distance asc;
lm = limit sort 1;
generate flatten(group) as (id), flatten(lm.tweet) as
tweet, flatten(lm.city_name) as city;
};
```

http://pig.apache.org/docs/r0.11.1/basic.html#nestedblock

PARALLEL Clause

- PARALLEL sets the number of reduce tasks for the MapReduce jobs generated by Pig. The default value is 1 (one reduce task).
- PARALLEL only affects the number of reduce tasks. Map parallelism is determined by the input file, one map for each HDFS block.
- If you don't specify PARALLEL, you still get the same map parallelism but only one reduce task.

```
a = load '/user/root/pig/full_text.txt' AS (id:chararray,
ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);
b = group a by id PARALLEL 18;
```

https://pig.apache.org/docs/r0.7.0/cookbook.html#Use+the+PARALLEL+Clause

Pig Functions

Pig Functions – Date/Time


```
---------------------------
-- DATE/Time functions
-- CurrentTime()
-- GetYear()
-- GetMonth()
-- GetDay()
-- GetWeek()
-- ToDate()
-- ToString()
-- ToUnixTime()
-------------------------


-- Date/Time functions
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float, tweet:chararray);
b = foreach a generate id, ts, ToDate(ts) as ts1;
c = foreach b generate id, ts, ts1, ToString(ts1) as ts_iso, ToUnixTime(ts1), GetYear(ts1) as year, GetMonth(ts1) as month,
GetWeek(ts1) as week;
d = limit c 5;
dump d;
```

DEMO

Date/Time Function Demo


```
grunt> a = load '/user/lab/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float, tweet:chararray);
grunt> b = foreach a generate id, ts, ToDate(ts) as ts1;
grunt> c = foreach b generate id, ts, ts1, ToString(ts1) as ts_iso, ToUnixTime(ts1), GetYear(ts1) as year, GetMonth(ts1) as month, GetWeek(ts1) as week;
grunt> d = limit c 5;
grunt> dump d;
2015-03-09 19:18:31,207 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: LIMIT
2015-03-09 19:18:31,377 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2015-03-09 19:18:31,501 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, Const
ptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatten, PushUpFilter, Sp
2015-03-09 19:18:31,566 [main] INFO  org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for a: $2, $3, $4, $5
2015-03-09 19:18:32,599 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2015-03-09 19:18:32,770 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2015-03-09 19:18:32,780 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
2015-03-09 19:18:33,456 [main] INFO  org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task 'attempt__0001_m_000001_1' to hdfs
2/tmp-1176900911/_temporary/0/task__0001_m_000001
2015-03-09 19:18:33,518 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2015-03-09 19:18:33,534 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2015-03-09 19:18:33,534 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(USER_79321756,2010-03-03T04:15:26,2010-03-03T04:15:26.000Z,2010-03-03T04:15:26.000Z,1267589726,2010,3,9)
(USER_79321756,2010-03-03T04:55:32,2010-03-03T04:55:32.000Z,2010-03-03T04:55:32.000Z,1267592132,2010,3,9)
(USER_79321756,2010-03-03T05:13:34,2010-03-03T05:13:34.000Z,2010-03-03T05:13:34.000Z,1267593214,2010,3,9)
(USER_79321756,2010-03-03T05:28:02,2010-03-03T05:28:02.000Z,2010-03-03T05:28:02.000Z,1267594082,2010,3,9)
(USER_79321756,2010-03-03T05:56:13,2010-03-03T05:56:13.000Z,2010-03-03T05:56:13.000Z,1267595773,2010,3,9)
```

Pig Functions - String


```
-- Use SUBSTRING() to extract year from ts string
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);
b = foreach a generate id, ts, SUBSTRING(ts, 0,4);
c = limit b 5;
dump c;

-- Find first twitter handles mentioned in a tweet using regex_extract() function
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);
b = foreach a generate id, ts, location, LOWER(tweet) as tweet;
c = foreach b generate id, ts, location, REGEX_EXTRACT(tweet, '(.*)@user_(\\S{8})([: | ])(.*)',2) as tweet;
d = limit c 5;
dump d;

-- Finding users who tweet long tweets
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float,
tweet:chararray);
b = foreach a generate id, ts, location, SIZE(REPLACE(tweet, '@USER_\\w{8}', '') ) as tweet_len;
c = order b by tweet_len desc;
d = limit c 10;
dump d;
```

DEMO

String Function Demo


```
grunt> a = load '/user/lab/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:float, lon:float, tweet:chararray);
grunt> b = foreach a generate id, ts, location, SIZE(REPLACE(tweet, '@USER_\\w{8}', '') ) as tweet_len;
grunt> c = order b by tweet_len desc;
grunt> d = limit c 10;
grunt> dump d;


(USER_2f88c77e,2010-03-06T16:57:30,?T: 36.888035,-76.227592,158)
(USER_3ca302f1,2010-03-06T01:49:43,?T: 30.019751,-95.444788,151)
(USER_3ca302f1,2010-03-04T01:15:19,?T: 30.019751,-95.444788,150)
(USER_2c8d1305,2010-03-05T18:48:13,?T: 33.994123,-118.280561,149)
(USER_c8613ca2,2010-03-04T17:20:27,?T: 40.623387,-73.917237,147)
(USER_4cecd527,2010-03-05T03:22:45,?T: 47.573729,-122.312608,146)
(USER_11ac7eaf,2010-03-03T20:02:46,?T: 30.452975,-91.108623,146)
(USER_77ee1910,2010-03-04T03:01:28,?T: 39.950974,-75.166685,146)
(USER_ae406f1d,2010-03-03T17:25:55,Pre: 40.729685,-74.006611,146)
(USER_2dcd8488,2010-03-06T21:37:02,iPhone: 30.426498,-91.137184,146)
```

UDF – User Defined Function

- Java
  - Java has the most extensive support. You can customize all parts of the processing including:
- Jython/Python/Javascript/Ruby/Groovy
  - Limited support is provided for Python, Jython etc.
- Where to get UDFs?
  - Built-In Functions
  - Piggybank/DataFu/Pigeon/etc.
    - Download from project site (maven central)
    - Download source and compile

UDF Usage

- Find the JAR online via maven central
  - OR download source and build your own in Eclipse
- Upload the JAR to Hadoop Sandbox
- In Pig
  - Register the JAR first
  - Define the functions to use
  - Invoke the function

Piggybank

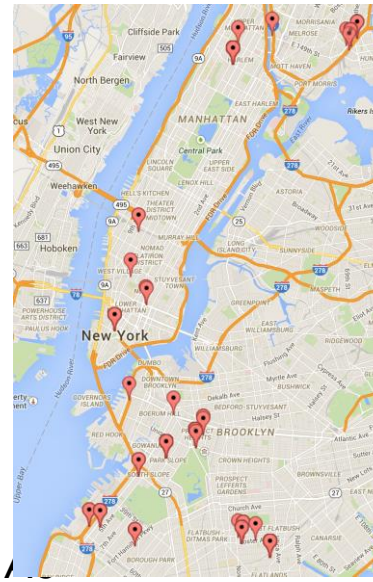
http://search.maven.org


```
[root@sandbox ~]# wget
http://central.maven.org/maven2/org/apache/pig/piggybank/0.14.0/piggybank-
0.14.0.jar

-----------------------

-- piggybank UDFs

-----------------------

register '/root/lab/piggybank-0.14.0.jar';

define isotounix org.apache.pig.piggybank.evaluation.datetime.convert.ISOToUnix();


a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray,
lat:float, lon:float, tweet:chararray);

b = foreach a generate id, ts, isotounix(ts) as ts_unix;

c = limit b 3;

dump c;
```

Pigeon UDF Demo

Find tweets tweeted from NYC


```
[root@sandbox ~]# wget https://github.com/Esri/geometry-api-java/releases/download/v1.2.1/esri-geometry-api-1.2.1.jar
[root@sandbox ~]# wget http://spatialhadoop.cs.umn.edu/pigeon/pigeon-0.1.jar
```




```
-- register UDFs
register /root/lab/pigeon-0.1.jar;
register /root/lab/esri-geometry-api-1.2.1.jar;
-- define functions
DEFINE ST_MakeBox edu.umn.cs.pigeon.MakeBox;
DEFINE ST_Contains edu.umn.cs.pigeon.Contains;
DEFINE ST_MakePoint edu.umn.cs.pigeon.MakePoint;

data = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray, location:chararray, lat:double, lon:double, tweet:chararray);
data1 = FOREACH data GENERATE id, ts, lat, lon, ST_MakePoint(lat, lon) AS geom_point, tweet;
data2 = FILTER data1 BY ST_Contains(ST_MakeBox(40.4774, -74.2589, 40.9176, -73.7004), geom_point);
data3 = limit data2 200;
data4 = foreach data3 generate lat, lon;
dump data4;
```

http://www.darrinward.com/lat-long/?id=490564

Scalar Projections

- Pig allows you to cast the elements of a single-tuple relation into a scalar value
- The primary use case for casting relations to scalars is the ability to use the values of global aggregates in follow up computations

Scalar Projection Demo

Normalize average number of tweets per user


```
a = load '/user/root/pig/full_text.txt' AS (id:chararray, ts:chararray,
location:chararray, lat:float, lon:float, tweet:chararray);

b = group a by id;

c = foreach b generate group as id, COUNT(a) as user_cnt;


d = group c ALL;

e = foreach d generate AVG(c.user_cnt) as global_avg;


f = foreach c generate id, user_cnt/(float)e.global_avg as index;


store f into '/user/root/pig/tweet_count_index';
```

id	cnt
user1	45
user2	22
user3	67

	global_avg
ALL	44.67

id	cnt	index
user1	45	**1.00**
user2	22	**0.49**
user3	67	**1.50**