# Advanced Pig

Monday, March 07, 2011
2:25 PM

Advanced Pig (or "we're not in Kansas anymore")
- Set operations in Map/Reduce
- How to parameterize an operation
- The oxymoron called "Pig Efficiency"

## Set operations

Monday, March 07, 2011
2:28 PM

Set operations in Map/Reduce

- UNION is easy
- Intersection and Difference rely upon COGROUP operation

COGROUP

Monday, March 07, 2011
2:35 PM

## COGROUP

Input: two bags with a pair of comparable columns.
Output: One bag in which the two bags are grouped together by the column.
Sort of "half of a join".

Example of COGROUP
if s1 has schema {name:chararray, hits:int} and data

```
(John,3)
(Harry,4)
(George,2)
```

and s2 has schema {name:chararray, errors:int} and data

```
(John,2)
(John,3)
(George,0)
(Sue,1)
```

Then

```
foo = COGROUP s1 BY name, s2 BY
name;
```

has schema {group:chararray, s1:{(name:chararray, hits:int)}, s2:{(name: chararray, errors:int)}} and returns

```
(John,{(John,
3)},{(John,2),(John,3)})
(Harry,{(Harry,4)},{})
(George,{(George,2)},{(George,0)})
(Sue, {},{(Sue,1)})
```

Note: Something is in the intersection of s1 and s2 if there are no {}'s in the cogroup.
Something is in the difference between s1 and s2 if there are no non-empty second sets.

## Set intersection

Monday, March 07, 2011
2:45 PM

Set intersection

If we have s1:{thing:chararray} and s2:{thing:chararray}

then we can form their intersection via COGROUP and FILTER

Example:

```
grps = COGROUP s1 BY thing, s2 BY thing;
-- cogroup by common
grp2 = FILTER grps by NOT(IsEmpty(s1)) AND NOT(IsEmpty(s2));
-- throw away non-compliant things
inter = FOREACH grp2 GENERATE group as thing;
-- strip the co-group
```

## Set difference

Monday, March 07, 2011
2:48 PM

Set difference

Use COGROUP to determine whether sets are empty or not.

USE FILTER to strike elements that are present in the second set:

Example: if s1:{thing:chararray} and s2:{thing:chararray} then

```
grps = COGROUP s1 BY thing, s2 BY thing;
-- it's in the difference if it is in the LHS, but not in the RHS
grp2 = FILTER grps by IsEmpty(s2);
diff = FOREACH grp2 GENERATE group as thing;
```

## Parameters and CROSS

Monday, March 07, 2011
2:51 PM

There are no true variables in Pig.

- Often, we want to set a parameter, e.g., how many things constitute a threshold.
- We can't do this in the script itself.

How do we parameterize an operation?

- Example: want to be able to change the number of friends someone should have in order to "count" in a query.
- Step 1: store the parameter in a file.
- Step 2: distribute the parameter via CROSS
- Step 3: do the distributed operation.
- Step 4: strip the distributed parameter.

## CROSS

Monday, March 07, 2011
2:52 PM

Suppose s1 and s2 are bags. Then

```
foo = CROSS s1,s2
```

contains all tuples built from one tuple in s1 and another in s2

Example: if foo contains

```
(1,2)
(3,4)
```

and bar contains

```
(amy,fred)
(george, jack)
```

then CROSS foo,bar contains

```
(1,2,amy,fred)
(1,2,george,jack)
(3,4,amy,fred)
(3,4,george,jack)
```

## Example: parameterize an attribute of a FILTER

Monday, March 07, 2011
3:00 PM

Suppose we have a relation friends: {name:chararray, friend:chararray}

```
(Amy,George)
(George,Fred)
(Fred, Anne)
(George,Joe)
(George,Harry)
```

and want to select people who have a certain number of friends.

We create a parameters file 'params.dat' containing

```
nfriends 2
```

and load it via

```
params = LOAD  'params.dat' USING PigStorage(' ') AS
(name:chararray, value:int);
```

Then we group the pairs by first friend

```
groups = GROUP friends BY name;
```

to get:

```
(Amy,{(Amy,George)})
(Fred, {(Fred, Anne)})
(George,{(George,Joe),(George,Harry),(George,Fred)})
```

and count the number of friends

```
group2 = FOREACH groups GENERATE group as name,
COUNT(friends) as count;
```

to get

```
(Amy, 1L)
(Fred, 1L)
(George, 3L)
```

Now we need to filter by the number of friends.
We select the parameter of interest:

```
nfriends = FILTER params BY name=='nfriends';
nfriend2 = FOREACH nfriends GENERATE value;
```

This results in the relation nfriend2: (value: int) containing

```
(2)
```

Now we CROSS that relation with the group2 relation

```
group3 = CROSS group2, nfriend2;
```

to get

```
(Amy, 1L, 2)
(Fred, 1L, 2)
(George, 3L, 2)
```

And finally, filter by the parameter

```
group4 = FILTER group3 by
group2::count>=nfriend2::value;
```

to get

```
(George, 3L, 2)
```

After which we can strip the parameter from the row:

```
group5 = FOREACH group4 GENERATE group2::name
AS name, group2::count AS count;
```

To get the one tuple we want, i.e.,

```
(George, 3L)
```

## Pig "Efficiency"

Monday, March 07, 2011
3:13 PM

- What is "Pig Efficiency"?
  - Pig script takes 1.7x java program time to do the same thing
  - Contributions include:
    - need to distribute data and code.
    - dynamically, as computation progresses
  - Runtime in Pig is affected by
    - Data size: how much you have to deal with.
    - Data distribution: is data you need where you need it?
- How to write "efficient" Pig scripts:
  - FILTER as early as possible.
  - PROJECT out useless attributes as early as possible.
  - minimize Map/Reduce phases.

Dirty Pig tricks

Monday, March 07, 2011
4:48 PM

## Dirty Pig Tricks

- canonicalization: if you have lots of variants of a thing, choose one
  - A symmetric relation (s1,s2) in R => (s2,s1) in R.
  - FILTER r by s1<s2; -- canonicalizes the pairs
- parameter distribution via CROSS/flatten
- you can move FILTERs upwards from the bottom, without changing the output of the script.
  - s = SOMETHING(r);
  - t = FILTER s BY s1<s2;
- can be reversed.