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

Set operations in Map/Reduce
UNION is easy
Intersection and Difference rely upon COGROUP operation
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 $s_1$ and $s_2$ if there are no {}'s in the cogroup.

Something is in the difference between $s_1$ and $s_2$ if there are no non-empty second sets.
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

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

\[ \text{grps} = \text{COGROUP s1 BY thing, s2 BY thing;} \]
\[ \text{-- it's in the difference if it is in the LHS, but not in the RHS} \]
\[ \text{grp2} = \text{FILTER grps by IsEmpty(s2);} \]
\[ \text{diff} = \text{FOREACH grp2 GENERATE group as thing;} \]
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.
Suppose $s_1$ and $s_2$ are bags. Then

$$\text{foo} = \text{CROSS } s_1, s_2$$

contains all tuples built from one tuple in $s_1$ and another in $s_2$

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)
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:

\[
\text{nfriends} = \text{FILTER params BY name=='nfriends'};
\]

\[
\text{nfriend2} = \text{FOREACH nfriends GENERATE value};
\]

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

(2)

Now we CROSS that relation with the group2 relation

\[
\text{group3} = \text{CROSS group2, nfriend2};
\]

to get

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

And finally, filter by the parameter

\[
\text{group4} = \text{FILTER group3 by group2::count}>=\text{nfriend2::value};
\]

to get

(George, 3L, 2)

After which we can strip the parameter from the row:

\[
\text{group5} = \text{FOREACH group4 GENERATE group2::name AS name, group2::count AS count};
\]

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

(George, 3L)
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

canonicalization: if you have lots of variants of a thing, choose one
  A symmetric relation \((s_1, s_2)\) in \(R\) \(\Rightarrow\) \((s_2, s_1)\) in \(R\).
  FILTER \(r\) by \(s_1 < s_2\); -- canonicalizes the pairs
  parameter distribution via CROSS/flatten
  you can move FILTERs upwards from the bottom,
  without changing the output of the script.
  \(s = \text{SOMETHING}(r)\);
  \(t = \text{FILTER } s \text{ BY } s_1 < s_2\);
  can be reversed.