MAP-REDUCE ABSTRACTIONS

Abstractions On Top Of Hadoop

- We've decomposed some algorithms into a map-reduce "workflow" (series of map-reduce steps)
  - naive Bayes training
  - naïve Bayes testing
  - phrase scoring
- How else can we express these sorts of computations? Are there some common special cases of map-reduce steps we can parameterize and reuse?

Abstractions On Top Of Hadoop

- Some obvious streaming processes:
  - for each row in a table
    - Transform it and output the result
    - Decide if you want to keep it with some boolean test, and copy out only the ones that pass the test

Example: stem words in a stream of word-count pairs:

("aardvarks",1) ➔ ("aardvark",1)

Proposed syntax: *f(row)* ➔ *row'*

table2 = MAP *table1* TO λ *row* : *f*(*row*))

Example: apply stop words

("aardvark",1) ➔ ("aardvark",1)
("the",1) ➔ *deleted*

Proposed syntax: *f(row)* ➔ {*true*,*false*}

table2 = FILTER *table1* BY λ *row* : *f*(*row*))

Abstractions On Top Of Hadoop

- A non-obvious? streaming processes:
  - for each row in a table
    - Transform it to a list of items
    - Splice all the lists together to get the output table (**flatten**)

Proposed syntax: *f(row)* ➔ *list of rows*

table2 = FLATMAP *table1* TO λ *row* : *f*(*row*))

Example: tokenizing a line

"I found an aardvark" ➔ ["i", "found","an","aardvark"]
"We love zymurgy" ➔ ["we","love","zymurgy"]

..but final table is one word per row

"i"
"found"
"an"
"aardvark"
"we"
"love"
...

Abstractions On Top Of Hadoop

- Another example from the Naïve Bayes test program…

NB Test Step (Can we do better?)

How:

- Stream and sort:
  - for each C[X=w^Y=y]=n
    - print "w C[Y=y]=n"
  - sort and build a *list* of values associated with each key *w*

  Like an inverted index

Event counts

$X=w_1$^$Y=sports$	*5245*
$X=w_1$^$Y=worldNews$	*1054*
X=..	*2120*
$X=w_2$^$Y=…$	*37*
X=…	*3*
…	…

w	Counts associated with W
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worldNews]=564
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldNews]=4464

NB Test Step 1 (Can we do better?)

The general case:
We're taking rows from a table
- In a particular format *(event,count)*

Applying a function to get a new value
- The *word* for the event

And *grouping* the rows of the table by this new value

➔ **Grouping operation**
Special case of a map-reduce

Event counts

$X=w_1 \wedge Y=sports$	5245
$X=w_1 \wedge Y=worldNews$	1054
$X=..$	2120
$X=w_2 \wedge Y=…$	37
$X=…$	3
…	…

Proposed syntax: *f(row)* ➔ *field*

GROUP *table* BY λ *row* : *f*(*row*)

Could define *f* via: a function, a field of a defined *record* structure, …

w	Counts associated with W
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worldNews]=564
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldNews]=4464

NB Test Step 1 (Can we do better?)

The general case:
We're taking rows from a table
- In a particular format *(event,count)*

Applying a function to get a new value
- The *word* for the event

And *grouping* the rows of the table by this new value

➔ **Grouping operation**
Special case of a map-reduce

Proposed syntax: *f(row)* ➔ *field*

GROUP *table* BY λ *row* : *f*(*row*)

Could define *f* via: a function, a field of a defined *record* structure, …

Aside: you guys know how to implement this, right?

1. Output pairs (f(row),row) with a map/streaming process
2. Sort pairs by key – which is f(row)
3. Reduce and aggregate by *appending together* all the values associated with the same key

Abstractions On Top Of Hadoop

- And another example from the Naïve Bayes test program…

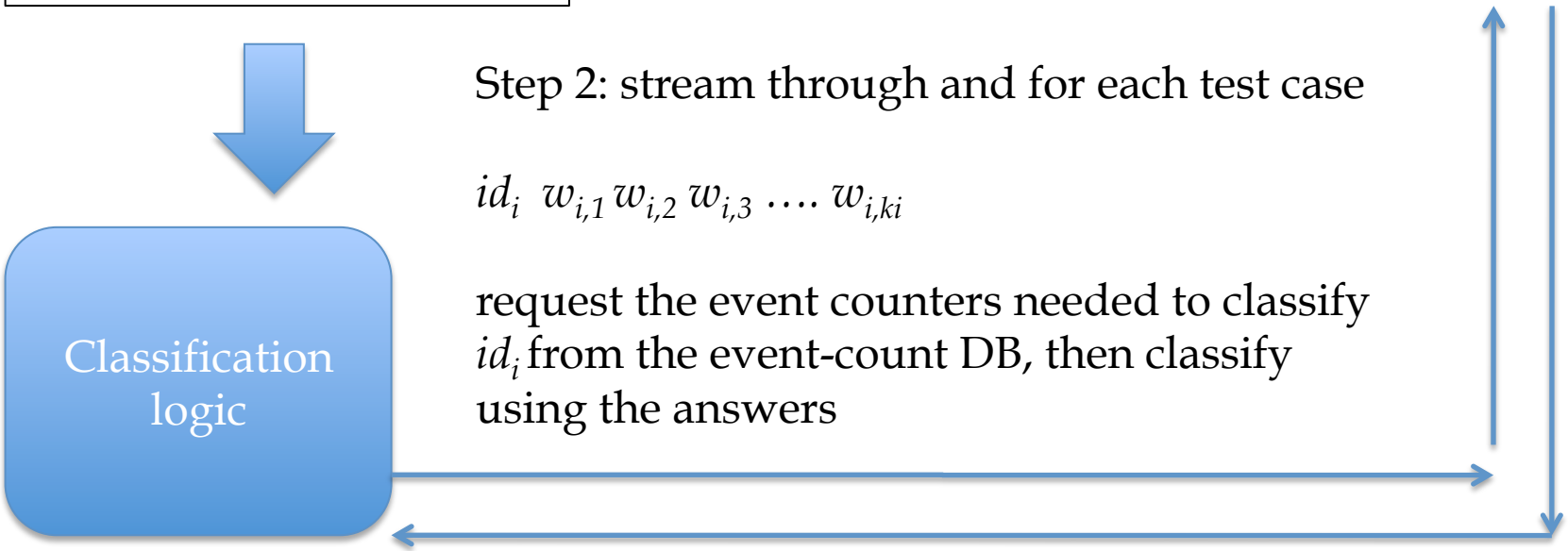
Request-and-answer

Test data

id_1 $w_{1,1}$ $w_{1,2}$ $w_{1,3}$ …. $w_{1,k1}$
id_2 $w_{2,1}$ $w_{2,2}$ $w_{2,3}$ ….
id_3 $w_{3,1}$ $w_{3,2}$ ….
id_4 $w_{4,1}$ $w_{4,2}$ …
id_5 $w_{5,1}$ $w_{5,2}$ ….
..

Record of all event counts for each word

w	Counts associated with W
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worl
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldN



Request-and-answer

- Break down into stages
  - Generate the data being requested (indexed by key, here a word)
    - Eg with group … by
  - Generate the requests as (key, requestor) pairs
    - Eg with flatmap … to
  - **Join** these two tables by key
    - Join defined as (1) cross-product and (2) filter out pairs with different values for keys
    - This replaces the step of concatenating two different tables of key-value pairs, and reducing them together
  - Postprocess the joined result

w	Request
found	~ctr to id1
aardvark	~ctr to id1
…	
zynga	~ctr to id1
…	~ctr to id2

w	Counters
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worldNews
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldNews]=

w	Counters	Requests
aardvark	C[w^Y=sports]=2	~ctr to id1
agent	C[w^Y=sports]=…	~ctr to id345
agent	C[w^Y=sports]=…	~ctr to id9854
agent	C[w^Y=sports]=…	~ctr to id345
…	C[w^Y=sports]=…	~ctr to id34742
zynga	C[…]	~ctr to id1
zynga	C[…]	…

w	Request
found	id1
aardvark	id1
…	
zynga	id1
…	id2

w	Counters
aardvark	C[w^Y=sports]=2
agent	C[w^Y=sports]=1027,C[w^Y=worldNews
…	…
zynga	C[w^Y=sports]=21,C[w^Y=worldNews]=

w	Counters	Requests
aardvark	C[w^Y=sports]=2	id1
agent	C[w^Y=sports]=…	id345
agent	C[w^Y=sports]=…	id9854
agent	C[w^Y=sports]=…	id345
…	C[w^Y=sports]=…	id34742
zynga	C[…]	id1
zynga	C[…]	…

IMPLEMENTATIONS OF MAP-REDUCE ABSTRACTIONS

Hive and PIG: word count

- Declarative ….. Fairly stable

```
FROM
(MAP docs.contents USING 'tokenizer_script' AS word, cnt
FROM docs
CLUSTER BY word) map_output

REDUCE map_output.word, map_output.cnt USING 'count_script' AS word, cnt;
```

```
A = load '/tmp/bible+shakes.nopunc';
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;
C = filter B by word matches '\w+';
D = group C by word;
E = foreach D generate COUNT(C) as count, group as word;
F = order E by count desc;
store F into '/tmp/wc';
```

PIG program is a bunch of **assignments** where every LHS is a **relation**.
No loops, conditionals, etc allowed.

More on Pig

- Pig Latin
  - atomic types + compound types like tuple, bag, map
  - execute locally/interactively or on hadoop
- can embed Pig in Java (and Python and …)
- can call out to Java from Pig
- Similar (ish) system from Microsoft: **DryadLinq**

```
A = load '/tmp/bible+shakes.nopunc';
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;
C = filter B by word matches '\w+';
D = group C by word;
E = foreach D generate COUNT(C) as count, group as word;
F = order E by count desc;
store F into '/tmp/wc';
```

Tokenize – built-in function

Flatten – special keyword, which applies to the next step in the process – ie foreach is transformed from a MAP to a FLATMAP

PIG parses and **optimizes** a sequence of commands before it executes them
It's smart enough to turn GROUP … FOREACH… SUM … into a map-reduce

- LOAD '*hdfs-path'* AS (*schema)*
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE *…* AS …, *…*
  - *transforms each row of a relation*
- DESCRIBE *alias/* ILLUSTRATE *alias -- debugging*
- GROUP *alias* BY *…*
- FOREACH *alias* GENERATE *group,* SUM(….)
  - *GROUP/GENERATE … aggregate op together act like a map-reduce*
- JOIN *r* BY *field, s* BY *field, …*
  - *inner join to produce rows: r::f1, r::f2, … s::f1, s::f2, …*
- CROSS *r, s, …*
  - *use with care unless all but one of the relations are singleton*
- User defined functions as operators
  - *also for loading, aggregates, …*

Phrase Finding in PIG

Phrase Finding I - loading the input

```
wcohen@shell2:~/pigtrial$ pig
2014-04-01 16:38:07,694 [main] INFO  org.apache.pig.Main - Apache Pig version 0.11.1.1.3.0.0-107 (rexported) compiled
2014-04-01 16:38:07,695 [main] INFO  org.apache.pig.Main - Logging error messages to: /h/wcohen/pigtrial/pig_1396384
2014-04-01 16:38:07,826 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /h/wcohen/.pigbootup not fo
2014-04-01 16:38:08,133 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to h
2014-04-01 16:38:08,379 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to m
grunt> SET default_parallel 10;
SET default_parallel 10;
grunt> fs -ls phrases/data/dkos-phraseFreq-5/
fs -ls phrases/data/dkos-phraseFreq-5/
Found 5 items
-rwxr-xr-x   3 wcohen supergroup      28857 2014-03-14 14:00 /user/wcohen/phrases/data/dkos-phraseFreq-5/part-00000
-rwxr-xr-x   3 wcohen supergroup      28210 2014-03-14 14:00 /user/wcohen/phrases/data/dkos-phraseFreq-5/part-00001
-rwxr-xr-x   3 wcohen supergroup      29731 2014-03-14 14:00 /user/wcohen/phrases/data/dkos-phraseFreq-5/part-00002
-rwxr-xr-x   3 wcohen supergroup      27422 2014-03-14 14:00 /user/wcohen/phrases/data/dkos-phraseFreq-5/part-00003
-rwxr-xr-x   3 wcohen supergroup      29198 2014-03-14 14:00 /user/wcohen/phrases/data/dkos-phraseFreq-5/part-00004
grunt> fs -tail phrases/data/dkos-phraseFreq-5/part-00003
fs -tail phrases/data/dkos-phraseFreq-5/part-00003
oluntary code   1.0
volvodrivingliberal sun 1.0
voreddy thu     1.0
voter registrations     2.0
voter suppression       3.0
wackyguy thu    1.0
waitingtoderail tue     1.0
walt starr      1.0
walter reed     1.0
wanna run       1.0
war plans       1.0
war question    1.0
war veterans    1.0

      ...

years taken     1.0
yes men 1.0
yesterday got   1.0
yesterday senator       1.0
yesterdays diary        1.0
york political  1.0
young people    1.0
zogby poll      1.0
zombiexx thu    1.0
```


years taken 1.0
yes men 1.0
yesterday got 1.0
yesterday senator 1.0
yesterdays diary 1.0
york political 1.0
young people 1.0
zogby poll 1.0
zombiexx thu 1.0

PIG Features

- comments -- *like this /* or like this */*
- 'shell-like' commands:
  - fs -ls … -- *any hadoop fs … command*
  - some shorter cuts: *ls, cp, …*
  - sh ls -al -- *escape to shell*

```
grunt> fgPhrases1 = LOAD 'phrases/data/dkos-phraseFreq-5/' AS (xy,c:int);
fgPhrases1 = LOAD 'phrases/data/dkos-phraseFreq-5/' AS (xy,c:int);
grunt> fgPhrases = FOREACH fgPhrases1 GENERATE STRSPLIT(xy,' ') AS xy:(x,y)
fgPhrases = FOREACH fgPhrases1 GENERATE STRSPLIT(xy,' ') AS xy:(x,y), c AS
2014-04-01 16:42:44,881 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:42:44,881 [main] WARN  org.apache.pig.PigServer - Encountered
grunt> DESCRIBE fgPhrases;
DESCRIBE fgPhrases;
2014-04-01 16:43:06,631 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:43:06,631 [main] WARN  org.apache.pig.PigServer - Encountered
fgPhrases: {xy: (x: bytearray,y: bytearray),c: int}
grunt> ILLUSTRATE fgPhrases;
     ...
-------------------------------------------------
| fgPhrases1     | xy:bytearray    | c:int    |
-------------------------------------------------
|                | patachon mon    | 1        |
-------------------------------------------------
-----------------------------------------------------------------------
| fgPhrases     | xy:tuple(x:bytearray,y:bytearray)          | c:int    |
-----------------------------------------------------------------------
|               | (patachon, mon)                            | 1        |
-----------------------------------------------------------------------
```

PIG Features

- comments -- *like this /* or like this */*
- 'shell-like' commands:
  - fs -ls … -- *any hadoop fs … command*
  - some shorter cuts: *ls, cp, …*
  - sh ls -al -- *escape to shell*
- LOAD '*hdfs-path*' AS (*schema*)
  - *schemas can include int, double, …*
  - *schemas can include complex types: bag, map, tuple, …*
- FOREACH *alias* GENERATE … AS …, …
  - *transforms each row of a relation*
  - *operators include +, -, and, or, …*
  - *can extend this set easily (more later)*
- DESCRIBE *alias -- shows the schema*
- ILLUSTRATE *alias -- derives a sample tuple*

Phrase Finding I - word counts

```
grunt> bgPhrases1 = LOAD 'phrases/data/brown-phraseFreq-5/' AS (xy,c:int);
bgPhrases1 = LOAD 'phrases/data/brown-phraseFreq-5/' AS (xy,c:int);
2014-04-01 16:46:52,014 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 1 time(s).
2014-04-01 16:46:52,014 [main] WARN  org.apache.pig.PigServer - Encountered Warning USING_OVERLOADED_FUNCTION 1 time(s).
grunt> bgPhrases = FOREACH bgPhrases1 GENERATE STRSPLIT(xy,' ') AS xy:(x,y), c AS c;
bgPhrases = FOREACH bgPhrases1 GENERATE STRSPLIT(xy,' ') AS xy:(x,y), c AS c;
2014-04-01 16:46:54,750 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 2 time(s).
2014-04-01 16:46:54,750 [main] WARN  org.apache.pig.PigServer - Encountered Warning USING_OVERLOADED_FUNCTION 2 time(s).
grunt> fgWordFreq1 = GROUP fgPhrases BY xy.x;
fgWordFreq1 = GROUP fgPhrases BY xy.x;
-- compute word frequencies

fgWordFreq1 = GROUP fgPhrases BY xy.x;
fgWordFreq = FOREACH fgWordFreq1 GENERATE group as w,SUM(fgPhrases.c) as c;
```

fgPhrases1	xy:bytearray	c:int
	expressly gave	1
	expressly reasserted	1

fgPhrases	xy:tuple(x:bytearray,y:bytearray)	c:int
	(expressly, gave)	1
	(expressly, reasserted)	1

fgWordFreq1	group:bytearray	fgPhrases:bag{:tuple(xy:tuple(x:bytearray,y:bytearray),c:int)}
	expressly	{((expressly, gave), 1), ((expressly, reasserted), 1)}

fgWordFreq	w:bytearray	c:long
	expressly	2

PIG Features

- LOAD '*hdfs-path*' AS (*schema*)
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE … AS …, …
  - *transforms each row of a relation*
- DESCRIBE *alias*/ ILLUSTRATE *alias* -- *debugging*
- GROUP *r* BY *x*
  - *like a shuffle-sort: produces relation with fields group and r, where r is a bag*

fgWordFreq1	group:bytearray	fgPhrases:bag{:tuple(xy:tuple(x:bytearray,y:bytearray),c:int)}
	expressly	{((expressly, gave), 1), ((expressly, reasserted), 1)}

PIG parses and **optimizes** a sequence of commands before it executes them
It's smart enough to turn GROUP … FOREACH… SUM … into a map-reduce

```
-- compute word frequencies

fgWordFreq1 = GROUP fgPhrases BY xy.x;
fgWordFreq = FOREACH fgWordFreq1 GENERATE group as w,SUM(fgPhrases.c) as c;
```

```
-------------------------------------------------
| fgPhrases1     | xy:bytearray         | c:int     |
-------------------------------------------------
|                | expressly gave       | 1         |
|                | expressly reasserted | 1         |
-------------------------------------------------
```

```
-------------------------------------------------------------------------
| fgPhrases     | xy:tuple(x:bytearray,y:bytearray)          | c:int     |
-------------------------------------------------------------------------
|               | (expressly, gave)                          | 1         |
|               | (expressly, reasserted)                    | 1         |
-------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------------------------
| fgWordFreq1     | group:bytearray     | fgPhrases:bag{:tuple(xy:tuple(x:bytearray,y:bytearray),c:int)}
-------------------------------------------------------------------------------------------------------------
|                 | expressly           | {((expressly, gave), 1), ((expressly, reasserted), 1)}
-------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------
| fgWordFreq     | w:bytearray     | c:long     |
-----------------------------------------
|                | expressly       | 2          |
-----------------------------------------
```

PIG Features

- LOAD '*hdfs-path*' AS (*schema)*
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE *…* AS …, *…*
  - *transforms each row of a relation*
- DESCRIBE *alias/* ILLUSTRATE *alias -- debugging*
- GROUP *alias* BY *…*
- FOREACH *alias* GENERATE *group*, SUM(….)
  - *GROUP/GENERATE … aggregate op together act like a map-reduce*
  - *aggregates: COUNT, SUM, AVERAGE, MAX, MIN, …*
  - *you can write your own*

PIG parses and **optimizes** a sequence of commands before it executes them
It's smart enough to turn GROUP … FOREACH… SUM … into a map-reduce

```
-- compute word frequencies

fgWordFreq1 = GROUP fgPhrases BY xy.x;
fgWordFreq = FOREACH fgWordFreq1 GENERATE group as w,SUM(fgPhrases.c) as c;



bgWordFreq1 = GROUP bgPhrases BY xy.x;
bgWordFreq = FOREACH bgWordFreq1 GENERATE group as w,SUM(bgPhrases.c) as c;
-- STORE bgWordFreq INTO 'phrases/data/bgWordFreq';
```

Phrase Finding 3 - assembling phrase- and word-level statistics

```
-- join in phrase stats, and then clean up
phraseStats1 = JOIN fgPhrases BY xy, bgPhrases BY xy;
phraseStats2 = FOREACH phraseStats1
                 GENERATE fgPhrases::xy AS xy, fgPhrases::c AS fC, bgPhrases::c AS bC;

-- join in word freqs for x and clean up
phraseStats3 = JOIN fgWordFreq BY w, bgWordFreq BY w, phraseStats2 by xy.x;
phraseStats4 = FOREACH phraseStats3
                 GENERATE xy,fC,bC,fgWordFreq::c as fxC,bgWordFreq::c as bxC;

-- join in word freqs for y and clean up
phraseStats5 = JOIN fgWordFreq BY w, bgWordFreq BY w, phraseStats4 by xy.y;
phraseStats6 = FOREACH phraseStats5
                 GENERATE xy,fC,bC,fxC,bxC,fgWordFreq::c as fyC,bgWordFreq::c as byC;
```

```
phraseStats1: {fgPhrases::xy: (x: bytearray,y: bytearray),fgPhrases::c: int,
               bgPhrases::xy: (x: bytearray,y: bytearray),bgPhrases::c: int}
```

bgWordFreq1	group:bytearray	bgPhrases:bag{:tuple(xy:tuple(x:bytearray,y:bytearray),c:int)}
	friday	{((friday, afternoon), 1)}
	afternoon	{((afternoon, service), 1), ((afternoon, mando), 1)}

bgWordFreq	w:bytearray	c:long
	friday	1
	afternoon	2

phraseStats1	fgPhrases::xy:tuple(x:bytearray,y:bytearray)	fgPhrases::c:int	bgPhrases::xy:tuple(x:bytearray,y:bytearray)	bgPhrases::c:int
	(friday, afternoon)	1	(friday, afternoon)	1

phraseStats2	xy:tuple(x:bytearray,y:bytearray)	fC:int	bC:int
	(friday, afternoon)	1	1

phraseStats3	fgWordFreq::w:bytearray	fgWordFreq::c:long	bgWordFreq::w:bytearray	bgWordFreq::c:long	phraseStats2::xy:tuple(x:bytearray,y:bytearray)	phraseStats2::fC:int	phraseStats2::bC:int
	friday	2	friday	1	(friday, afternoon)	1	1

phraseStats4	phraseStats2::xy:tuple(x:bytearray,y:bytearray)	phraseStats2::fC:int	phraseStats2::bC:int	fxC:long	bxC:long
	(friday, afternoon)	1	1	2	1

...

phraseStats6	phraseStats4::phraseStats2::xy:tuple(x:bytearray,y:bytearray)	phraseStats4::phraseStats2::fC:int	phraseStats4::phraseStats2::bC:int	phraseStats4::fxC:long	phraseStats4::bxC:long	fyC:long	byC:long
	(friday, afternoon)	1	1	2	1	1	2

bgWordFreq1	group:bytearray	bgPhrases:bag{:tuple(xy:tuple(x:bytearray,y:bytearray),c:int)}
	friday	{((friday, afternoon), 1)}
	afternoon	{((afternoon, service), 1), ((afternoon, mando), 1)}

bgWordFreq	w:bytearray	c:long
	friday	1
	afternoon	2

phraseStats1	fgPhrases::xy:tuple(x:bytearray,y:bytearray)	fgPhrases::c:int	bgPhrases::xy:tuple(x:bytearray,y:bytearray)	bgPhrase
	(friday, afternoon)	1	(friday, afternoon)	1

phraseStats2	xy:tuple(x:bytearray,y:bytearray)	fC:int	bC:int
	(friday, afternoon)	1	1

phraseStats3	fgWordFreq::w:bytearray	fgWordFreq::c:long	bgWordFreq::w:bytearray	bgWordFreq::c:long	phraseStats2::xy:tuple(x:bytearray,y:bytea
	friday	2	friday	1	(friday, afternoon)

phraseStats4	phraseStats2::xy:tuple(x:bytearray,y:bytearray)	phraseStats2::fC:int	phraseStats2::bC:int	fxC:long	bxC:long
	(friday, afternoon)	1	1	2	1

PIG Features

- LOAD '*hdfs-path*' AS (*schema)*
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE *…* AS …, *…*
  - *transforms each row of a relation*
- DESCRIBE *alias/* ILLUSTRATE *alias -- debugging*
- GROUP *alias* BY *…*
- FOREACH *alias* GENERATE *group,* SUM(….)
  - *GROUP/GENERATE … aggregate op together act like a map-reduce*
- JOIN *r* BY *field, s* BY *field, …*
  - inner join to produce rows: *r::f1, r::f2, … s::f1, s::f2, …*

```
phraseStats1: {fgPhrases::xy: (x: bytearray,y: bytearray),fgPhrases::c: int,
               bgPhrases::xy: (x: bytearray,y: bytearray),bgPhrases::c: int}
```

Phrase Finding 4 - adding total frequencies

```
grunt> fgPhraseCount1 = group fgPhrases1 ALL;
fgPhraseCount1 = group fgPhrases1 ALL;
2014-04-01 16:57:31,934 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:57:31,934 [main] WARN  org.apache.pig.PigServer - Encountered
grunt> fgPhraseCount = FOREACH fgPhraseCount1 GENERATE SUM(fgPhrases1.c);
fgPhraseCount = FOREACH fgPhraseCount1 GENERATE SUM(fgPhrases1.c);
2014-04-01 16:57:34,607 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:57:34,607 [main] WARN  org.apache.pig.PigServer - Encountered
grunt> bgPhraseCount1 = group bgPhrases1 ALL;
bgPhraseCount1 = group bgPhrases1 ALL;
2014-04-01 16:57:38,271 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:57:38,271 [main] WARN  org.apache.pig.PigServer - Encountered
grunt> bgPhraseCount = FOREACH bgPhraseCount1 GENERATE SUM(bgPhrases1.c);
bgPhraseCount = FOREACH bgPhraseCount1 GENERATE SUM(bgPhrases1.c);
2014-04-01 16:57:40,577 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:57:40,577 [main] WARN  org.apache.pig.PigServer - Encountered

-------------------------------------------------
| bgPhrases1     | xy:bytearray      | c:int      |
-------------------------------------------------
|                | continuing series | 1          |
|                | neighboring lower | 1          |
-------------------------------------------------

-----------------------------------------------------------------------------------------------------------
| bgPhraseCount1     | group:chararray      | bgPhrases1:bag{:tuple(xy:bytearray,c:int)}                |
-----------------------------------------------------------------------------------------------------------
|                    | all                  | {(continuing series, 1), (neighboring lower, 1)}          |
-----------------------------------------------------------------------------------------------------------

-----------------------------------
| bgPhraseCount     | :long      |
-----------------------------------
|                   | 2          |
-----------------------------------
```

How do we add the totals to the phraseStats relation?

```
grunt> counts1 = CROSS fgPhraseCount,bgPhraseCount;
counts1 = CROSS fgPhraseCount,bgPhraseCount;
2014-04-01 16:59:38,370 [main] WARN  org.apache.pig.PigServer - E
2014-04-01 16:59:38,370 [main] WARN  org.apache.pig.PigServer - E
grunt> counts = FOREACH counts1 GENERATE $0 AS fTot,$1 as bTot;
counts = FOREACH counts1 GENERATE $0 AS fTot,$1 as bTot;
2014-04-01 16:59:42,024 [main] WARN  org.apache.pig.PigServer - E
2014-04-01 16:59:42,024 [main] WARN  org.apache.pig.PigServer - E
grunt> phraseStats = CROSS phraseStats6,counts;
phraseStats = CROSS phraseStats6,counts;
2014-04-01 16:59:45,083 [main] WARN  org.apache.pig.PigServer - E
2014-04-01 16:59:45,083 [main] WARN  org.apache.pig.PigServer - E
grunt> STORE phraseStats INTO 'phrases/data/phraseStats';
```

STORE triggers execution of the query plan….

it also limits optimization

```
fs -tail phrases/data/phraseStats/part-r-00001

(preliminary,data)      1       1       2       16      4       39      9194    99888
(best,way)      1       5       15      164     3       53      9194    99888
(tour,reached)  1       1       1       3       1       25      9194    99888
(right,way)     1       1       25      85      3       53      9194    99888
(cold,war)      1       19      1       60      16      53      9194    99888
(long,way)      1       10      9       291     3       53      9194    99888
(best,book)     1       1       15      164     3       31      9194    99888
(receive,new)   1       1       4       20      81      1083    9194    99888
(just,got)      6       2       68      258     14      68      9194    99888
(really,got)    1       1       19      148     14      68      9194    99888
(phone,calls)   1       1       7       9       1       11      9194    99888
(congressional,offices) 1       1       7       12      1       4       9194    99888
(second,major)  1       3       11      193     20      182     9194    99888
(special,events)        1       2       6       163     1       12      9194    99888
(civil,rights)  2       5       11      59      6       6       9194    99888
(managing,editor)       1       1       1       2       1       8       9194    99888
(national,press)        1       1       41      255     23      41      9194    99888
(associated,press)      3       1       3       9       23      41      9194    99888
(senate,foreign)        3       2       18      26      7       98      9194    99888
(law,clerk)     1       1       5       47      1       5       9194    99888
(making,clear)  1       1       7       75      7       30      9194    99888
(mutual,fund)   1       1       1       23      2       14      9194    99888
(court,justices)        1       1       21      74      2       2       9194    99888
(sharp,contrast)        1       2       1       41      1       4       9194    99888
(foreign,policy)        1       18      7       98      3       31      9194    99888
```

Comment: schema is lost when you store….

PIG Features

- LOAD '*hdfs-path*' AS (*schema)*
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE *…* AS …, *…*
  - *transforms each row of a relation*
- DESCRIBE *alias/* ILLUSTRATE *alias -- debugging*
- GROUP *alias* BY *…*
- FOREACH *alias* GENERATE *group,* SUM(….)
  - *GROUP/GENERATE … aggregate op together act like a map-reduce*
- JOIN *r* BY *field, s* BY *field, …*
  - *inner join to produce rows: r::f1, r::f2, … s::f1, s::f2, …*
- CROSS *r, s, …*
  - *use with care unless all but one of the relations are singleton*
  - *newer pigs allow singleton relation to be* ***cast to a scalar***

Phrase Finding 5 - phrasiness and informativeness

How do we compute some complicated function?

With a "UDF"

```
package com.wcohen;

import java.io.*;
import java.util.*;

import org.apache.pig.*;
import org.apache.pig.data.*;
import org.apache.pig.impl.util.WrappedIOException;


public class SmoothedPKL extends EvalFunc<Double>
{
    public static double smoothPKL(double k1,double n1,double k2,double n2,double p0,double m) {
        return PKL(k1 + p0*m, n1+m, k2+p0*m, n2+m);
    }
    public static double PKL(double k1,double n1,double k2,double n2) {
        double p1 = k1/n1;
        double p2 = k2/n2;
        return p1 * Math.log(p1/p2);
    }

    @Override
    public Double exec(Tuple input) throws IOException {
        if (input==null || input.size()!=6) { return null; }
        double k1,n1,k2,n2,p0,m;
        try {
            k1 = DataType.toDouble(input.get(0));
            n1 = DataType.toDouble(input.get(1));
            k2 = DataType.toDouble(input.get(2));
            n2 = DataType.toDouble(input.get(3));
            p0 = DataType.toDouble(input.get(4));
            m  = DataType.toDouble(input.get(5));
        } catch (Exception e) {
            throw WrappedIOException.wrap("Error in Phrases processing row ",e);
        }
        return smoothPKL(k1,n1,k2,n2,p0,m);
    }

}
```

```
phraseStats = LOAD 'phrases/data/phraseStats' AS (xy:(x,y),fC,bC,fxC,bxC,fyC,byC,fTot,bTot);

-- final compute phraseness, etc

REGISTER ./pkl.jar;

phraseResult = FOREACH phraseStats GENERATE *,
             com.wcohen.SmoothedPKL(fC, fTot, bC, bTot, 1.0/bTot, 1.0) as infoness,
             com.wcohen.SmoothedPKL(fC, fTot, fxC*fyC, fTot*fTot, 1.0/fxC, 1.0) as phraseness;

STORE phraseResult INTO 'phrases/data/phraseResult';
```

PIG Features

- LOAD '*hdfs-path*' AS (*schema)*
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE *…* AS …, *…*
  - *transforms each row of a relation*
- DESCRIBE *alias/* ILLUSTRATE *alias -- debugging*
- GROUP *alias* BY *…*
- FOREACH *alias* GENERATE *group,* SUM(….)
  - *GROUP/GENERATE … aggregate op together act like a map-reduce*
- JOIN *r* BY *field, s* BY *field, …*
  - *inner join to produce rows: r::f1, r::f2, … s::f1, s::f2, …*
- CROSS *r, s, …*
  - *use with care unless all but one of the relations are singleton*
- User defined functions as operators
  - *also for loading, aggregates, …*

The full phrase-finding pipeline

```
-- load data
fgPhrases1 = LOAD 'phrases/data/dkos-phraseFreq-5/' AS (xy,c:int);
fgPhrases = FOREACH fgPhrases1 GENERATE STRSPLIT(xy,' ') AS xy:(x,y), c AS c;
bgPhrases1 = LOAD 'phrases/data/brown-phraseFreq-5/' AS (xy,c:int);
bgPhrases = FOREACH bgPhrases1 GENERATE STRSPLIT(xy,' ') AS xy:(x,y), c AS c;

-- compute word frequencies
fgWordFreq1 = GROUP fgPhrases BY xy.x;
fgWordFreq = FOREACH fgWordFreq1 GENERATE group as w,SUM(fgPhrases.c) as c;
bgWordFreq1 = GROUP bgPhrases BY xy.x;
bgWordFreq = FOREACH bgWordFreq1 GENERATE group as w,SUM(bgPhrases.c) as c;

-- join in phrase stats, and then clean up schema
phraseStats1 = JOIN fgPhrases BY xy, bgPhrases BY xy;
STORE phraseStats1 INTO 'phrases/data/phraseStats1';
phraseStats2 = FOREACH phraseStats1 GENERATE fgPhrases::xy AS xy, fgPhrases::c AS fC, bgPhrases::c AS bC;
-- join in word freqs for x and clean up
phraseStats3 = JOIN fgWordFreq BY w, bgWordFreq BY w, phraseStats2 by xy.x;
phraseStats4 = FOREACH phraseStats3 GENERATE xy,fC,bC,fgWordFreq::c as fxC,bgWordFreq::c as bxC;
-- join in word freqs for y and clean up
phraseStats5 = JOIN fgWordFreq BY w, bgWordFreq BY w, phraseStats4 by xy.y;
phraseStats6 = FOREACH phraseStats5 GENERATE xy,fC,bC,fxC,bxC,fgWordFreq::c as fyC,bgWordFreq::c as byC;

-- compute totals
fgPhraseCount1 = group fgPhrases1 ALL;
fgPhraseCount = FOREACH fgPhraseCount1 GENERATE SUM(fgPhrases1.c);
bgPhraseCount1 = group bgPhrases1 ALL;
bgPhraseCount = FOREACH bgPhraseCount1 GENERATE SUM(bgPhrases1.c);

-- join in totals - ok to use cross-product here since all but one table are just singletons
counts1 = CROSS fgPhraseCount,bgPhraseCount;
counts = FOREACH counts1 GENERATE $0 AS fTot,$1 as bTot;
phraseStats = CROSS phraseStats6,counts;

-- finally compute phraseness, etc

REGISTER ./pkl.jar;
phraseResult = FOREACH phraseStats GENERATE *,
            com.wcohen.SmoothedPKL(fC, fTot, bC, bTot, 1.0/bTot, 1.0) as infoness,
            com.wcohen.SmoothedPKL(fC, fTot, fxC*fyC, fTot*fTot, 1.0/fxC, 1.0) as phraseness;
STORE phraseResult INTO 'phrases/data/phraseResult';
```