PigMix

Pig Mix

PigMix is a set of queries used test pig performance from release to release. There are queries that test latency (how long does it take to run this query?), and queries that test scalability (how many fields or records can pig handle before it fails?). In addition it includes a set of map reduce java programs to run equivalent map reduce jobs directly. These will be used to test the performance gap between direct use of map reduce and using pig. In Jun 2010, we release PigMix2, which include 5 more queries in addition to the original 12 queries into PigMix to measure the performance of new Pig features. We will publish the result of both PigMix and PigMix2.

Usage

To run PigMix, run the following command from PIG_HOME:

```
ant -Dharness.hadoop.home=$HADOOP_HOME pigmix-deploy (generate test dataset)
ant -Dharness.hadoop.home=$HADOOP_HOME pigmix (run the PigMix benchmark)
```

You can optionally set HADOOP_CONF_DIR before run.

If you want to change the default size of test dataset, change test/perf/pigmix/conf/config.sh.

Note the PigMix is checked in to Pig 0.12 and beyond. If you want to run it in earlier version of Pig, Please go to https://issues.apache.org/jira/browse/PIG-200 and use PIG-200-0.12.patch.

Runs

PigMix

The following table includes runs done of the pig mix. All of these runs have been done on a cluster with 26 slaves plus one machine acting as the name node and job tracker. The cluster was running hadoop version 0.18.1. (TODO: Need to get specific hardware info on those machines).

The tests were run against two versions of pig: top of trunk, and top of types branch both as of Nov 21 2008.

The tests were run three times for each version and the results averaged.

tot = top of trunk
totb = top of types branch

Version	Map Reduce Java Code	tot 11/21 /08	totb 11/21 /08	totb 1/20 /09	tot 2/23 /09
Date Run	11/22/08	11/21/08	11/21/08	1/20/09	2/23/09
L1 explode	116	261	283	218	205
L2 fr join	41	1665	253	168	89
L3 join	97	1912	320	258	254
L4 distinct agg	68	254	193	110	116
L5 anti-join	90	1535	281	209	112
L6 large group by key	61	294	226	126	120
L7 nested split	72	243	204	107	102
L8 group all	56	462	194	104	103
L9 order by 1 field	286	5294	867	851	444
L10 order by multiple fields	634	1403	565	469	447
L11 distinct + union	120	316	255	164	154
L12 multi-store	150	fails	781	499	804
Total time	1791	13638	4420	3284	2950
Compared to hadoop	1.0	7.6	2.5	1.8	1.6
Weighted Average	1.0	11.2	3.26	2.20	1.97

The totb run of 1/20/09 includes the change to make BufferedPositionedInputStream use a buffer instead of relying on hadoop to buffer.

tot run of 2/23/09, top of trunk is now what was on the types branch (that is proto 0.2.0). This run includes fragment replicate join and rework of partitioning for order by.

Run of 5/28/09, placed in a separate table because there were underlying cluster changes, thus the map reduce tests needed to be rerun. This is the same code base that became 0.3.0.

Version	Map Reduce Java code	tot 5/27 /09
Date Run	5/28/09	5/28/09
L1 explode	119	205
L2 fr join	44	110
L3 join	113	314
L4 distinct agg	76	153
L5 anti-join	96	128
L6 large group by key	67	148
L7 nested split	67	133
L8 group all	64	115
L9 order by 1 field	329	563
L10 order by multiple fields	607	532
L11 distinct + union	106	203
L12 multi-store	139	159
Total time	1826	2764
Compared to hadoop	N/A	1.5
Weighted average	N/A	1.83

Run date: June 28, 2009, run against top of trunk as of that day.
Note that the columns got reversed in this one (Pig then MR)

Test	Pig run time	Java run time	Multiplier
PigMix_1	204	117.33	1.74
PigMix_2	110.33	50.67	2.18
PigMix_3	292.33	125	2.34
PigMix_4	149.67	85.33	1.75
PigMix_5	131.33	105	1.25
PigMix_6	146.33	65.33	2.24
PigMix_7	128.33	82	1.57
PigMix_8	126.33	63.67	1.98
PigMix_9	506.67	312.67	1.62
PigMix_10	555	643	0.86
PigMix_11	206.33	136.67	1.51
PigMix_12	173	161.67	1.07
Total	2729.67	1948.33	1.40
Weighted avg			1.68

Run date: August 27, 2009, run against top of trunk as of that day.

Test	Pig run time	Java run time	Multiplier
PigMix_1	218	133.33	1.635
PigMix_2	99.333	48	2.07
PigMix_3	272	127.67	2.13
PigMix_4	142.33	76.333	1.87
PigMix_5	127.33	107.33	1.19
PigMix_6	135.67	73	1.86
PigMix_7	124.67	78.333	1.59
PigMix_8	117.33	68	1.73
PigMix_9	356.33	323.67	1.10
PigMix_10	511.67	684.33	0.75

PigMix_11	180	121	1.49
PigMix_12	156	160.67	0.97
Total	2440.67	2001.67	1.22
Weighted avg			1.53

Run date: October 18, 2009, run against top of trunk as of that day.
With this run we included a new measure, weighted average. Our previous multiplier that we have been publishing takes the total time of running all 12 Pig Latin scripts and compares it to the total time of running all 12 Java Map Reduce programs. This is a valid way to measure, as it shows the total amount of time to do all these operations on both platforms. But it has the drawback that it gives more weight to long running operations (such as joins and order bys) while masking the performance in faster operations such as group bys. The new "weighted average" adds up the multiplier for each Pig Latin script vs. Java program separately and then divides by 12, thus weighting each test equally. In past runs the weighted average had significantly lagged the overall average (for example, in the run above for August 27 it was 1.5 even though the total difference was 1.2). With this latest run it still lags some, but the gap has shrunk noticably.

Test	Pig run time	Java run time	Multiplier
PigMix_1	135.0	133.0	1.02
PigMix_2	46.67	39.33	1.19
PigMix_3	184.0	98.0	1.88
PigMix_4	71.67	77.67	0.92
PigMix_5	70.0	83.0	0.84
PigMix_6	76.67	61.0	1.26
PigMix_7	71.67	61.0	1.17
PigMix_8	43.33	47.67	0.91
PigMix_9	184.0	209.33	0.88
PigMix_10	268.67	283.0	0.95
PigMix_11	145.33	168.67	0.86
PigMix_12	55.33	95.33	0.58
Total	1352.33	1357	1.00
Weighted avg			1.04

Run date: January 4, 2010, run against 0.6 branch as of that day

Test	Pig run time	Java run time	Multiplier
PigMix_1	138.33	112.67	1.23
PigMix_2	66.33	39.33	1.69
PigMix_3	199	83.33	2.39
PigMix_4	59	60.67	0.97
PigMix_5	80.33	113.67	0.71
PigMix_6	65	77.67	0.84
PigMix_7	63.33	61	1.04
PigMix_8	40	47.67	0.84
PigMix_9	214	215.67	0.99
PigMix_10	284.67	284.33	1.00
PigMix_11	141.33	151.33	0.93
PigMix_12	55.67	115	0.48
Total	1407	1362.33	1.03
Weighted Avg			1.09

PigMix2

Run date: May 29, 2010, run against top of trunk as of that day.

Test	Pig run time	Java run time	Multiplier
PigMix_1	122.33	117	1.05
PigMix_2	50.33	42.67	1.18

PigMix_3	189	100.33	1.88
PigMix_4	75.67	61	1.24
PigMix_5	64	138.67	0.46
PigMix_6	65.67	69.33	0.95
PigMix_7	88.33	84.33	1.05
PigMix_8	39	47.67	0.82
PigMix_9	274.33	215.33	1.27
PigMix_10	333.33	311.33	1.07
PigMix_11	151.33	157	0.96
PigMix_12	70.67	97.67	0.72
PigMix_13	80	33	2.42
PigMix_14	69	86.33	0.80
PigMix_15	80.33	69.33	1.16
PigMix_16	82.33	69.33	1.19
PigMix_17	286	229.33	1.25
Total	2121.67	1929.67	1.10
Weighted Avg			1.15

Run date: Jun 11, 2011, run against top of trunk as of that day.

Test	**Pig run time**	**Java run time**	**Multiplier**
PigMix_1	130	139	0.94
PigMix_2	66	48.67	1.36
PigMix_3	138	107.33	1.29
PigMix_4	106	78.33	1.35
PigMix_5	135.67	114	1.19
PigMix_6	103.67	74.33	1.39
PigMix_7	77.67	77.33	1.00
PigMix_8	56.33	57	0.99
PigMix_9	384.67	280.33	1.37
PigMix_10	380	354.67	1.07
PigMix_11	164	141	1.16
PigMix_12	109.67	187.33	0.59
PigMix_13	78	44.33	1.76
PigMix_14	105.33	111.67	0.94
PigMix_15	89.67	87	1.03
PigMix_16	87.67	75.33	1.16
PigMix_17	171.33	152.33	1.12
Total	2383.67	2130	1.12
Weighted Avg			1.16

Pig 0.9.2

Test	**Pig run time**	**Java run time**	**Multiplier**
PigMix_1	146	147	0.993197278911565
PigMix_2	73	61	1.19672131147541
PigMix_3	134	158	0.848101265822785
PigMix_4	91	87	1.04597701149425
PigMix_5	81	153	0.529411764705882
PigMix_6	91	81	1.12345679012346
PigMix_7	71	86	0.825581395348837

PigMix_8	56	61	0.918032786885246
PigMix_9	302	192	1.57291666666667
PigMix_10	312	226	1.38053097345133
PigMix_11	207	222	0.932432432432432
PigMix_12	96	163	0.588957055214724
PigMix_13	76	127	0.598425196850394
PigMix_14	94	157	0.598726114649682
PigMix_15	86	92	0.934782608695652
PigMix_16	80	82	0.975609756097561
PigMix_17	196	176	1.11363636363636
Total	2192	2271	0.965213562
Weighted Avg			0.951558634

Pig 0.10.1

Test	**Pig run time**	**Java run time**	**Multiplier**
PigMix_1	147	146	1.00684931506849
PigMix_2	74	62	1.19354838709677
PigMix_3	140	158	0.886075949367089
PigMix_4	87	86	1.01162790697674
PigMix_5	81	153	0.529411764705882
PigMix_6	92	262	0.351145038167939
PigMix_7	76	86	0.883720930232558
PigMix_8	62	61	1.01639344262295
PigMix_9	303	187	1.62032085561497
PigMix_10	303	232	1.30603448275862
PigMix_11	188	218	0.862385321100917
PigMix_12	101	157	0.643312101910828
PigMix_13	82	132	0.621212121212121
PigMix_14	99	158	0.626582278481013
PigMix_15	82	91	0.901098901098901
PigMix_16	82	82	1
PigMix_17	206	177	1.1638418079096
Total	2205	2448	0.900735294117647
Weighted Avg			0.919032977

Pig 0.11.1

Test	**Pig run time**	**Java run time**	**Multiplier**
PigMix_1	163	141	1.15602836879433
PigMix_2	66	61	1.08196721311475
PigMix_3	141	158	0.892405063291139
PigMix_4	87	86	1.01162790697674
PigMix_5	82	158	0.518987341772152

PigMix_6	92	81	1.1358024691358
PigMix_7	82	87	0.94252873563218 4
PigMix_8	63	62	1.01612903225806
PigMix_9	320	207	1.54589371980676
PigMix_10	311	226	1.37610619469027
PigMix_11	184	218	0.84403669724770 6
PigMix_12	97	158	0.61392405063291 1
PigMix_13	78	127	0.61417322834645 7
PigMix_14	101	158	0.63924050632911 4
PigMix_15	87	91	0.95604395604395 6
PigMix_16	82	87	0.94252873563218 4
PigMix_17	203	176	1.15340909090909
Total	2239	2282	0.98115687992988 6
Weighted Avg			0.967107783

Pig 0.12 (4/4/2013)

Test	**Pig run time**	**Java run time**	**Multiplier**
PigMix_1	168	142	1.1830985915493
PigMix_2	71	62	1.14516129032258
PigMix_3	141	158	0.89240506329113 9
PigMix_4	93	87	1.06896551724138
PigMix_5	87	158	0.55063291139240 5
PigMix_6	93	81	1.14814814814815
PigMix_7	77	87	0.88505747126436 8
PigMix_8	62	57	1.08771929824561
PigMix_9	310	192	1.61458333333333
PigMix_10	311	221	1.40723981900452
PigMix_11	190	217	0.87557603686635 9
PigMix_12	102	158	0.64556962025316 5
PigMix_13	77	133	0.57894736842105 3
PigMix_14	101	343	0.29446064139941 7
PigMix_15	87	86	1.01162790697674
PigMix_16	82	82	1
PigMix_17	207	177	1.16949152542373
Total	2259	2441	0.92544039328144 2
Weighted Avg			0.974040267

Features Tested

Based on a sample of user queries, PigMix includes tests for the following features.

1. Data with many fields, but only a few are used.
2. Reading data from maps.
3. Use of bincond and arithmetic operators.
4. Exploding nested data.
5. Load bzip2 data
6. Load uncompressed data

7. join with one table small enough to fit into a fragment and replicate algorithm.
8. join where tables are sorted and partitioned on the same key
9. Do a cogroup that is not immediately followed by a flatten (that is, use cogroup for something other than a straight forward join).
10. group by with only algebraic udfs that has nested plan (distinct aggs basically).
11. foreachs with nested plans including filter and implicit splits.
12. group by where the key accounts for a large portion of the record.
13. group all
14. union plus distinct
15. order by
16. multi-store query (that is, a query where data is scanned once, then split and grouped different ways).
17. outer join
18. merge join
19. multiple distinct aggregates
20. accumulative mode

The data is generated so that it has a zipf type distribution for the group by and join keys, as this models most human generated data.
Some other fields are generated using a uniform data distribution.

Scalability tests test the following:

1. Join of very large data sets.
2. Grouping of very large data set.
3. Query with a very wide (500+ fields) row.
4. Loading many data sets together in one load

Proposed Data

Initially, four data sets have been created. The first, "page_views", is 10 million rows in size, with a schema of:

Name	Type	Average Length	Cardinality	Distribution	Percent Null
user	string	20	1.6M	zipf	7
action	int	X	2	uniform	0
timespent	int	X	20	zipf	0
query_term	string	10	1.8M	zipf	20
ip_addr	long	X	1M	zipf	0
timestamp	long	X	86400	uniform	0
estimated_revenue	double	X	100k	zipf	5
page_info	map	15	X	zipf	0
page_links	bag of maps	50	X	zipf	20

The second, "users", was created by taking the unique user keys from "page_views" and adding additional columns.

Name	Type	Average Length	Cardinality	Distribution	Percent Null
name	string	20	1.6M	unique	7
phone	string	10	1.6M	zipf	20
address	string	20	1.6M	zipf	20
city	string	10	1.6M	zipf	20
state	string	2	1.6M	zipf	20
zip	int	X	1.6M	zipf	20

The third, "power_users", has 500 rows, and has the same schema as users. It was generated by skimming 500 unique names from users. This will produce a table that can be used to test fragment replicate type joins.

The fourth, "widerow", has a very wide row (500 fields), consisting of one string and 499 integers.

"users", "power_users", and "widerow" are written in ASCII format, using Ctrl-A as the field delimiter. They can be read using PigStorage.

"page_views" is written in as text data, with Ctrl-A as the field delimiter. Maps in the file are delimited by Ctrl-C between key value pairs and Ctrl-D between keys and values. Bags in the file are delimited by Ctrl-B between tuples in the bag. A special loader, PigPerformance loader has been written to read this format.

PigMix2 include 4 more data set, which can be derived from the original dataset:

```
A = load 'page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info, page_links);
B = order A by user $parallelfactor;
store B into 'page_views_sorted' using PigStorage('\u0001');

alpha = load 'users' using PigStorage('\u0001') as (name, phone, address, city, state, zip);
a1 = order alpha by name $parallelfactor;
store a1 into 'users_sorted' using PigStorage('\u0001');

a = load 'power_users' using PigStorage('\u0001') as (name, phone, address, city, state, zip);
b = sample a 0.5;
store b into 'power_users_samples' using PigStorage('\u0001');

A = load 'page_views' as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info,
page_links,
user as user1, action as action1, timespent as timespent1, query_term as query_term1, ip_addr as ip_addr1,
timestamp as timestamp1, estimated_revenue as estimated_revenue1, page_info as page_info1, page_links as
page_links1,
user as user2, action as action2, timespent as timespent2, query_term as query_term2, ip_addr as ip_addr2,
timestamp as timestamp2, estimated_revenue as estimated_revenue2, page_info as page_info2, page_links as
page_links2;
store B into 'widegroupbydata';
```

Proposed Scripts

Scalability

Script S1

This script tests grouping, projecting, udf envocation, and filtering with a very wide row. Covers scalability feature 3.

```
A = load '$widerow' using PigStorage('\u0001') as (name: chararray, c0: int, c1: int, ..., c500: int);
B = group A by name parallel $parrallelfactor;
C = foreach B generate group, SUM(A.c0) as c0, SUM(A.c1) as c1, ... SUM(A.c500) as c500;
D = filter C by c0 > 100 and c1 > 100 and c2 > 100 ... and c500 > 100;
store D into '$out';
```

Script S2
This script tests joining two inputs where a given value of the join key appears many times in both inputs. This will test pig's ability to handle large joins. It covers scalability features 1 and 2.

TBD

Features not yet tested: 4.

Latency

Script L1

This script tests reading from a map, flattening a bag of maps, and use of bincond (features 2, 3, and 4).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, (int)action as action, (map[])page_info as page_info,
    flatten((bag{tuple(map[])})page_links) as page_links;
C = foreach B generate user,
    (action == 1 ? page_info#'a' : page_links#'b') as header;
D = group C by user parallel 40;
E = foreach D generate group, COUNT(C) as cnt;
store E into 'L1out';
```

Script L2

This script tests using a join small enough to do in fragment and replicate (feature 7).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, estimated_revenue;
alpha = load '/user/pig/tests/data/pigmix/power_users' using PigStorage('\u0001') as (name, phone,
        address, city, state, zip);
beta = foreach alpha generate name;
C = join B by user, beta by name using 'replicated' parallel 40;
store C into 'L2out';
```

Script L3

This script tests a join too large for fragment and replicate. It also contains a join followed by a group by on the same key, something that pig could potentially optimize by not regrouping.

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, (double)estimated_revenue;
alpha = load '/user/pig/tests/data/pigmix/users' using PigStorage('\u0001') as (name, phone, address,
        city, state, zip);
beta = foreach alpha generate name;
C = join beta by name, B by user parallel 40;
D = group C by $0 parallel 40;
E = foreach D generate group, SUM(C.estimated_revenue);
store E into 'L3out';
```

Script L4

This script covers foreach generate with a nested distinct (feature 10).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, action;
C = group B by user parallel 40;
D = foreach C {
    aleph = B.action;
    beth = distinct aleph;
    generate group, COUNT(beth);
}
store D into 'L4out';
```

Script L5

This script does an anti-join. This is useful because it is a use of cogroup that is not a regular join (feature 9).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user;
alpha = load '/user/pig/tests/data/pigmix/users' using PigStorage('\u0001') as (name, phone, address,
        city, state, zip);
beta = foreach alpha generate name;
C = cogroup beta by name, B by user parallel 40;
D = filter C by COUNT(beta) == 0;
E = foreach D generate group;
store E into 'L5out';
```

Script L6

This script covers the case where the group by key is a significant percentage of the row (feature 12).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, action, (int)timespent as timespent, query_term, ip_addr, timestamp;
C = group B by (user, query_term, ip_addr, timestamp) parallel 40;
D = foreach C generate flatten(group), SUM(B.timespent);
store D into 'L6out';
```

Script L7

This script covers having a nested plan with splits (feature 11).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
as (user, action, timespent, query_term,
            ip_addr, timestamp, estimated_revenue, page_info, page_links);
B = foreach A generate user, timestamp;
C = group B by user parallel 40;
D = foreach C {
    morning = filter B by timestamp < 43200;
    afternoon = filter B by timestamp >= 43200;
    generate group, COUNT(morning), COUNT(afternoon);
}
store D into 'L7out';
```

Script L8

This script covers group all (feature 13).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, (int)timespent as timespent, (double)estimated_revenue as estimated_revenue;
C = group B all;
D = foreach C generate SUM(B.timespent), AVG(B.estimated_revenue);
store D into 'L8out';
```

Script L9

This script covers order by of a single value (feature 15).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = order A by query_term parallel 40;
store B into 'L9out';
```

Script L10

This script covers order by of multiple values (feature 15).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent:int, query_term, ip_addr, timestamp,
        estimated_revenue:double, page_info, page_links);
B = order A by query_term, estimated_revenue desc, timespent parallel 40;
store B into 'L10out';
```

Script L11

This script covers distinct and union and reading from a wide row but using only one field (features: 1, 14).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user;
C = distinct B parallel 40;
alpha = load '/user/pig/tests/data/pigmix/widerow' using PigStorage('\u0001');
beta = foreach alpha generate $0 as name;
gamma = distinct beta parallel 40;
D = union C, gamma;
E = distinct D parallel 40;
store E into 'L11out';
```

Script L12

This script covers multi-store queries (feature 16).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links);
B = foreach A generate user, action, (int)timespent as timespent, query_term,
    (double)estimated_revenue as estimated_revenue;
split B into C if user is not null, alpha if user is null;
split C into D if query_term is not null, aleph if query_term is null;
E = group D by user parallel 40;
F = foreach E generate group, MAX(D.estimated_revenue);
store F into 'highest_value_page_per_user';
beta = group alpha by query_term parallel 40;
gamma = foreach beta generate group, SUM(alpha.timespent);
store gamma into 'total_timespent_per_term';
beth = group aleph by action parallel 40;
gimel = foreach beth generate group, COUNT(aleph);
store gimel into 'queries_per_action';
```

Script L13 (PigMix2 only)

This script covers outer join (feature 17).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
        as (user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info, page_links);
B = foreach A generate user, estimated_revenue;
alpha = load '/user/pig/tests/data/pigmix/power_users_samples' using PigStorage('\u0001') as (name, phone,
address, city, state, zip);
beta = foreach alpha generate name, phone;
C = join B by user left outer, beta by name parallel 40;
store C into 'L13out';
```

Script L14 (PigMix2 only)

This script covers merge join (feature 18).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views_sorted' using org.apache.pig.test.udf.storefunc.
PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info, page_links);
B = foreach A generate user, estimated_revenue;
alpha = load '/user/pig/tests/data/pigmix/users_sorted' using PigStorage('\u0001') as (name, phone, address,
city, state, zip);
beta = foreach alpha generate name;
C = join B by user, beta by name using 'merge';
store C into 'L14out';
```

Script L15 (PigMix2 only)

This script covers multiple distinct aggregates (feature 19).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info, page_links);
B = foreach A generate user, action, estimated_revenue, timespent;
C = group B by user parallel 40;
D = foreach C {
    beth = distinct B.action;
    rev = distinct B.estimated_revenue;
    ts = distinct B.timespent;
    generate group, COUNT(beth), SUM(rev), (int)AVG(ts);
}
store D into 'L15out';
```

Script L16 (PigMix2 only)

This script covers accumulative mode (feature 20).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/page_views' using org.apache.pig.test.udf.storefunc.PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info, page_links);
B = foreach A generate user, estimated_revenue;
C = group B by user parallel 40;
D = foreach C {
    E = order B by estimated_revenue;
    F = E.estimated_revenue;
    generate group, SUM(F);
}

store D into 'L16out';
```

Script L17 (PigMix2 only)

This script covers wide key group (feature 12).

```
register pigperf.jar;
A = load '/user/pig/tests/data/pigmix/widegroupbydata' using org.apache.pig.test.udf.storefunc.
PigPerformanceLoader()
    as (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, page_info, page_links, user_1, action_1, timespent_1, query_term_1, ip_addr_1,
timestamp_1,
        estimated_revenue_1, page_info_1, page_links_1, user_2, action_2, timespent_2, query_term_2, ip_addr_2,
timestamp_2,
        estimated_revenue_2, page_info_2, page_links_2);
B = group A by (user, action, timespent, query_term, ip_addr, timestamp,
        estimated_revenue, user_1, action_1, timespent_1, query_term_1, ip_addr_1, timestamp_1,
        estimated_revenue_1, user_2, action_2, timespent_2, query_term_2, ip_addr_2, timestamp_2,
        estimated_revenue_2) parallel 40;
C = foreach B generate SUM(A.timespent), SUM(A.timespent_1), SUM(A.timespent_2), AVG(A.estimated_revenue), AVG
(A.estimated_revenue_1), AVG(A.estimated_revenue_2);
store C into 'L17out';
```

Features not yet covered: 5 (bzip data)

Data Generation

If you want to know the details of data generation, please see DataGeneratorHadoop.