Building the Brickhouse

Enhancing Hive with our UDF library
Data Pipeline in Hive

Advantages:
   Able to prototype quickly
   Extensible with UDFs

Disadvantages:
   Still need to understand "under the hood".
   Still "bleeding-edge"
   Enough rope to hang yourself
Solution: The Brickhouse

Generic UDF's to handle common situations
Design patterns and tools to deal
with "Big Data"
Approaches to improve performance/scalability
(Not just a bunch of functions)

Not necessarily only solution, but our solution.
Solution: The Brickhouse

Functionality centered along certain functional areas. Cookbook of "recipes" to solve certain general problems.

- collect
- distributed_cache
- sketch_set
- bloom
- json
- sanity
- hbase
- timeseries
Array/Map operations

collect
collect_max
cast_array
map_key_values
map_filter_keys
join_array
map_union
union_max
truncate_array
**collect**

Similar to Ruby/Scala collect (and Hive collect_set() )
UDAF aggregates multiple lines,
Returns map/array of values
Use with explode

```sql
select ks_uid,
   collect(dt),
   collect(score)
from
maxwell_score
group by ks_uid;
```

```sql
select ks_uid,
   collect(dt),
   collect(score)
from
maxwell_score
```
### collect

Opposite of UDTF
Avoids "self-join" Anti-pattern

```sql
select a.id,
a.value as a_val,
b.value as b_val
from ( select * from mytable where type='A' ) a
join
  (select * from mytable where type='B' ) b
on ( a.id = b.id );
```

```sql
select id,
col_map['A'] as a_val,
col_map['B'] as b_val
from
  ( select id,
collect( type,value)
  from mytable
  group by id )
```
**collect_max**

Similar to collect, but returns map with top 20 values. Utilize Hive map-side aggregation to reduce sort size.

```sql
select ks_uid,
    combined_score,
from maxwell_score
order by combined_score
limit 20;
```

```sql
select collect_max(ks_uid, combined_score )
from maxwell_score
where dt=20121008;
```
Salt your queries, and do in two steps, if your job is too big.

create table salty_aggs as
select ks_uid, random_salt,
    collect_max( actor_ks_uid, actor_klout_score)
     as top_score_map
from ( 
    select ks_uid,
        rand()*128 as random_salt,
        actor_ks_uid, actor_klout_score
    from big_table ) bt
group by ks_uid, random_salt;

select ks_uid,
    union_max( 
        top_score_map )
    as top_score_map
from salty_aggs
group by ks_uid;
distributed_map

Uses distributed-cache to access values in-memory.
Avoids join/resort of large datasets

```sql
select
  ks_uid
from big_table bt
join
  ( select *
      from celeb where
      is_celeb = true )
celeb
on
bt.ks_uid = celeb. ks_uid;

insert overwrite local directory 'celeb_map';

add file 'celeb_map';

select * from celeb
  where is_celeb = true;
add file celeb_map;

select * from big_table
  where
distributed_map( ks_uid, 'celeb_map' ) is not null;
```
multiday_count

Generates counts for 1, 3, 7, 21 days with one pass of the data.

```sql
select count(*),
       collect( actor_ks_uid )
from   action_table
where  dt <= today
       and dt > days_add(today, -7 )
union all
select count(*),
       collect( actor_ks_uid )
from   action_table
where  dt <= today
       and dt > days_add(today, -14)
union all...
```

```sql
select multiday_count(
    dt,cnt, actors, today,
    array( 1,3,7,30,60,90))
from action_table
where dt <= today
    and dt > today -90;
```
conditional_emit

Emit several different rows depending upon different conditions, in one pass

```sql
select ks_uid,
    'ALL' as feature_class
from user_table
union all
select ks_uid,
    'NY' as feature_class
from user_table
    where city = 'NY'
union all
select ks_uid, 'CELEB'
from user_table
    where is_celeb(ks_uid)
union all ...
```

```sql
select ks_uid
    conditional_emit(
        array( true,
            city = 'NY',
            is_celeb( ks_uid ) ),
        array('ALL','NY','CELEB')
    )
    as feature_class
from user_table;
```
Estimate number of uniques for large sets with a fixed amount of space.
KMV Sketch implementation.
Good for titans (@onedirection, @youtube)
Avoids "count distinct"

```sql
select
  count(distinct ks_uid) as reach
from
  actor_action
where
  some_condition() = true;
```

```sql
select estimated_reach(
  sketch_set( ks_uid ) )
from
  actor_action
where
  some_condition() = true;
```
**sketch_set**

Easy to do set unions.
Can aggregate incremental results.

```sql
insert overwrite table daily_sketch partition (dt=20130211) select sketch_set( ks_uid) ss from mytable;

select estimated_reach(
    union_sketch( ss) )
from daily_sketch
where dt >= days_add( today(), -30);
```
Algorithm:
Take MD5 Hash of your string.
Collect 5000 lowest hashes.
If set size < 5000 that is your reach.
If set size = 5000 use highest hash value to calculate reach
reach = 5000/(maxHash + MAX_LONG)
*2*MAX_LONG;
Why does it work ???
You need a very good hash
Why does it work ???
You need a very good hash.
MD5 will distribute hashes evenly.
Why does it work ???
You need a very good hash.
MD5 will distribute hashes evenly.

As number of hashes grows bigger, value of 5000th hash grows smaller.
bloom

bloom
bloom_contains
distributed_bloom
bloom_and
bloom_or
bloom_not
bloom

Currently uses HBase's bloomfilter implementation. Uses a large BitSet to express set membership. Can tell if set contains a key, but can't iterate over the keys.
bloom

Use similar to distributed_map.
Avoids a join and a re-sort.

```sql
select *
from
    content_items ci
left outer join
    deleted_content_items del
on
    ci.content_id = del.content_id
where del.content_id is null;
```

```sql
insert overwrite local directory 'del_items_bloom'
select bloom( content_id )
from deleted_content_item;
add file del_items_bloom;
select *
from content_items ci
left outer join
    deleted_content_items del
on
    ci.content_id = del.content_id
where del.content_id is null;
```
bloom

Can be merged easily for large sets.

```sql
insert overwrite local
directory 'thirty_day_bloom'
select bloom_and( bloom )
    from agg_bloom
    where dt >= days_add
        (today(), -30);

add file thirty_day_bloom;

select ks_uid
    from actor_action
where
bloom_contains(
distributed_bloom(
    'thirty_day_bloom' ));
```
assert, write_to_graphite

"Productionize" the pipeline.
Sanity checks for data quality.
Upload statistics to Graphite for visibility.

```sql
select write_to_graphite(grHost, grPort,
    "prod.maxwell.count.tw",count(*) ),
  assert(count(*) > 1000, "Low moment count.")
from hb_dash(moment);
```
to_json, from_json

Serialize to JSON
Avoid ugly, error-prone string concat's

select
concat("{"kscore":",
kscore, ","moving_avg":",
avg,
","start_date":",
start,
","end_date":",
end,"}"
)
from
mytable;

select
to_json(
    named_struct("kscore", 
    kscore, 
    "moving_avg",avg, 
    "start_date",start, 
    "end_date", end)  )
from mytable;
to_json, from_json

Serialize from JSON
Pass in a struct of the type to be returned

create view parse_json as
select
  ks_uid, from_json( json,
    named_struct("kscore", 0.0,
      "moving_avg", array( 0.0 ),
      "start_date", ",",
      "end_date", ","
    ) )
from moving_avg_view;
Alternative to HBase Handler
Distribute keys across HBase regions to balance load.
Uses Batch Puts

```
select ks_uid_salt,
    hbase_batch_put('my_hbase_table',
        ks_uid_key, hb_json, 500)
from hb_salted_view
distribute by ks_uid_salt;
```
Questions ???

Public Repo https://github.com/klout/brickhouse
Wiki/Documentation
https://github.com/klout/brickhouse/wiki

jerome@klout.com